(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平10－283199

(43)公開日　平成10年(1998)10月23日

(51)Int.Cl.[6]	識別記号	Ｆ Ｉ	
Ｇ０６Ｆ　9/46	３４０	Ｇ０６Ｆ　9/46	３４０Ａ

審査請求　未請求　請求項の数36　ＯＬ　外国語出願　（全158頁）

(21)出願番号　特願平9－180625

(22)出願日　平成９年(1997)６月20日

(31)優先権主張番号　０８／８２６，５６０

(32)優先日　1997年４月４日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　391055933
マイクロソフト　コーポレイション
ＭＩＣＲＯＳＯＦＴ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国　ワシントン州　98052－6399　レッドモンド　ワン　マイクロソフト　ウェイ　（番地なし）

(72)発明者　ジョージ　エイチ．　ジェイ．　ショウ
アメリカ合衆国　98072　ワシントン州　ウッドィンヴィル　ノースイースト　186　ティーエイチ　ストリート　21213

(74)代理人　弁理士　谷　義一　（外１名）

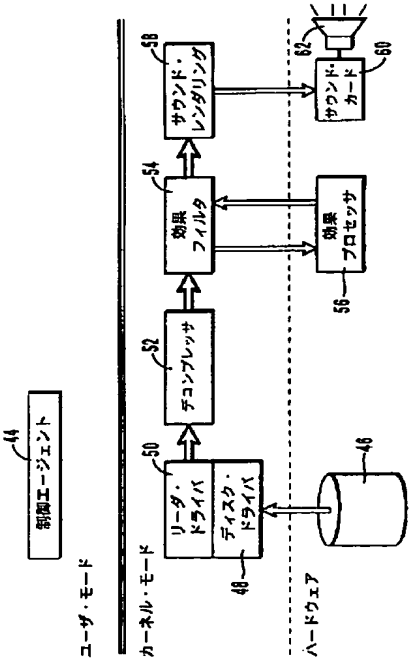
最終頁に続く

(54)【発明の名称】　標準クロック・メカニズムを用いて、多重データ・ストリームの処理の同期と異なる処理速度のマッチングを行う方法、コンピュータ・プログラム・プロダクツ、およびシステム

(57)【要約】　（修正有）

【課題】　カーネル・モードで実行される相互接続ソフトウェア・フィルタのシステムにおいて、２つまたはそれ以上のデータ・ストリーム間の処理を同期化し、相互に対してドリフトする可能性のある２つの異なるハードウェア・クロック間のレートを整合する。

【解決手段】　同期化はイベント通知またはストリーム位置の照会を通して達成され、別々のストリーム内のデータの対応するフレームが一緒にレンダリングされるようにする（例えば、ビデオ・フレームおよび対応するサウンド・トラック）。レート整合は物理クロックの進行を、一連のデータ・ストリーム・タイムスタンプと対比しながらモニタし、異なるクロック・レートを整合するように調整を行う。共通物理クロックは、あるコンポーネントが特定のクロック時間を、すべてのコンポーネントによって共有される時間に変換するときの参照として使用され、発生する誤差を最小限にする。



1

【特許請求の範囲】

【請求項１】　データをカーネル・モードで処理することを効率化するようにソフトウェア・ドライバを相互接続し、異なるデータ・ストリームを同期化または異なるハードウェア・クロックをレート整合するためのタイミング情報を標準化された方法で提供する方法であって、該方法は、

１つまたは２つ以上のカーネル・モード・ドライバをオープンするステップと、

ドライバを接続するための１つまたは２つ以上の接続ピン・インスタンスを形成するステップであって、各接続ピン・インスタンスは前記１つまたは２つ以上のドライバと階層的に関係づけられ、該１つまたは２つ以上のドライバ間でデータ伝送のために使用されるステップと、

レート整合とストリーム同期化のための１つまたは２つ以上のクロック・メカニズムを作成するステップであって、各クロックは、前記１つまたは２つ以上の接続ピン・インスタンスの１つと階層的に関係づけられ、基礎となるハードウェア・オシレータに基づいてデータ・ストリーム時間と物理時間を提供するステップと、

該１つまたは２つ以上の接続ピン・インスタンスを相互接続し、カーネル・モードに置かれた１つまたは２つ以上のドライバを通る連続データ・フロー・パスが得られるようにするステップであって、前記クロックは異種クロック・オシレーション間でレート整合するときと、異種データ・ストリームを同期化するときにタイミング情報を利用できるようにするステップとを備えていることを特徴とする方法。

【請求項２】　請求項１に記載の方法において、各接続ピン・インスタンスはファイル・オブジェクトによって表され、階層関係は、接続ピン・インスタンス・ファイル・オブジェクトの作成時に関係ドライバを指定することによって作成され、ドライバはシステム上で利用できる入出力デバイスのファイル・オブジェクトとして参照され、各クロック・メカニズムはファイル・オブジェクトによって表され、かつ、接続ピン・インスタンスとの階層関係はクロック・ファイル・オブジェクトの作成時に関係接続ピン・インスタンス・ファイル・オブジェクトを親として指定することによって作成されることを特徴とする方法。

【請求項３】　請求項１に記載の方法において、前記１つまたは２つ以上のドライバ上の前記接続ピン・インスタンスの各々について照会し、クロック・メカニズムが使用可能であるかどうかを判断し、特定の接続ピン・インスタンスで同期化またはレート整合を行うためのクロック・メカニズムをインスタンス生成すべきかどうかを判断してから、該１つまたは２つ以上の接続ピン・インスタンスを相互接続するようにするステップをさらに含むことを特徴とする方法。

【請求項４】　請求項１に記載の方法において、前記１つまたは２つ以上のドライバ上の前記１つまたは２つ以上の接続ピン・インスタンスの少なくとも１つは、少なくとも１つの事前定義されたプロパティ集合、メソッド集合、およびイベント集合をサポートし、前記少なくとも１つの接続ピン・インスタンスに対してクロック・メカニズムが利用可能であることをサード・パーティ・コンポーネントに示して、サード・パーティ・コンポーネントがそれぞれの接続ピン・インスタンスで前記クロック・メカニズムを作ることを可能にしており、クロックは少なくとも１つの事前定義されたプロパティ集合、メソッド集合、およびイベント集合をサポートして、前記それぞれのクロックをサード・パーティ・コンポーネントによって判断された通りに制御するようにしたことを特徴とする方法。

【請求項５】　コンピュータ・プログラム・プロダクトにおいて、

カーネル・モードの他のコンポーネントとの標準化された相互接続メカニズムを提供するためにそこに具現化されている、コンピュータ読取り可能プログラム・コード手段を収めているコンピュータ使用可能媒体であって、該コンピュータ使用可能媒体は、

接続ピン・インスタンスを形成するためのプログラム・コード手段であって、該接続ピン・インスタンスはカーネル・モードの他のコンポーネントとの間でデータを受け渡しするために使用され、他のコンポーネント上に置かれている他の接続ピン・インスタンスとの相互接続が可能になっている手段と、

該接続ピン・インスタンスのいくつかでクロック・メカニズムを形成するためのプログラム・コード手段であって、該クロック・メカニズムは時間インターバル情報が関連づけられているデータのストリーム内の現在位置を反映している位置時間値を提供する手段とを備えていることを特徴とするコンピュータ・プログラム・プロダクト。

【請求項６】　請求項５に記載のコンピュータ・プログラム・プロダクトにおいて、クロック・メカニズムは、複数のコンポーネントが利用できる共通ハードウェア・オシレータに基づく参照時間値と前記位置時間値とを備える相関時間値をアトミック・オペレーションでさらに提供し、他のコンポーネントが前記共通ハードウェア・オシレータを参照として使用して時間変換を行えるようにしたことを特徴とするコンピュータ・プログラム・プロダクト。

【請求項７】　請求項５に記載のコンピュータ・プログラム・プロダクトにおいて、クロック・メカニズムは、複数のコンポーネントが利用できる共通ハードウェア・オシレータに基づく参照時間値と指定時間値とを備える相関時間値をアトミック・オペレーションでさらに提供し、他のコンポーネントが前記共通ハードウェア・オシレータを参照として使用して時間変換を行えるようにし

たことを特徴とするコンピュータ・プログラム・プロダクト。

【請求項８】　請求項５に記載のコンピュータ・プログラム・プロダクトにおいて、クロック・メカニズムはハードウェア・オシレータに基づく物理時間値をさらに提供することを特徴とするコンピュータ・プログラム・プロダクト。

【請求項９】　請求項８に記載のコンピュータ・プログラム・プロダクトにおいて、クロック・メカニズムは、複数のコンポーネントが利用できる共通ハードウェア・オシレータに基づく参照時間値と前記物理時間値とを備える相関物理時間値をアトミック・オペレーションでさらに提供し、他のコンポーネントが前記共通ハードウェア・オシレータを参照として使用して時間変換を行えるようにしたことを特徴とするコンピュータ・プログラム・プロダクト。

【請求項１０】　請求項５に記載のコンピュータ・プログラム・プロダクトにおいて、前記位置時間値は前記プログラム・コード手段に実装されているプロパティ集合の一部として他のコンポーネントによってアクセスされることを特徴とするコンピュータ・プログラム・プロダクト。

【請求項１１】　請求項５に記載のコンピュータ・プログラム・プロダクトにおいて、クロック・メカニズムはさらに関係イベントの通知を他のコンポーネントに提供することを特徴とするコンピュータ・プログラム・プロダクト。

【請求項１２】　請求項５に記載のコンピュータ・プログラム・プロダクトにおいて、クロック・メカニズムはさらに関係イベントの通知を前記位置時間に基づいて他のコンポーネントに提供し、有用な同期点を提供するようにしたことを特徴とするコンピュータ・プログラム・プロダクト。

【請求項１３】　複数のデータ・ストリームを同期化する方法であって、

データ・サンプルの参照データ・ストリームに基づいてマスタ・クロック参照を作成するステップであって、各データ・サンプルは参照データ・ストリーム内の位置を示す時間インターバル情報をもち、マスタ・クロック参照は時間インターバル情報から作成されるステップと、

１つまたは２つ以上の他のデータ・ストリームをマスタ・クロック参照に基づいて処理し、他のデータ・ストリームが参照データ・ストリームの処理と同期化されるようにするステップとを備えることを特徴とする方法。

【請求項１４】　請求項１３に記載の方法において、マスタ・クロック参照はプロパティ集合の一部としてアクセスされることを特徴とする方法。

【請求項１５】　請求項１３に記載の方法において、マスタ・クロック参照は、サード・パーティ・エージェントからの要求に応答して作成される、選択的にインスタンス生成可能なクロック・メカニズムの一部であることを特徴とする方法。

【請求項１６】　請求項１３に記載の方法において、時間インターバル情報はメディア・サンプルに関するタイムスタンプ情報によって得られることを特徴とする方法。

【請求項１７】　請求項１３に記載の方法において、時間インターバル情報はメディア・ストリーム規則によって得られることを特徴とする方法。

【請求項１８】　処理コンポーネントを通して処理されるデータ・ストリームをハードウェア・プロセッサに合わせてレート整合する方法であって、

ハードウェア・プロセッサのハードウェア・オシレーションに基づく物理時間参照を処理コンポーネントに提供するステップと、

処理コンポーネントで行われるデータ・ストリームの処理レートを物理時間参照に基づいてハードウェア・プロセッサの処理レートに整合するように調整するステップであって、データ・ストリームは時間インターバル情報が関連づけられているサンプルから構成されているものとを備えることを特徴とする方法。

【請求項１９】　請求項１８に記載の方法において、物理時間参照はプロパティ集合の一部としてアクセスされることを特徴とする方法。

【請求項２０】　請求項１８に記載の方法において、物理時間参照は、サード・パーティ・エージェントからの要求に応答して作成される、選択的にインスタンス生成可能なクロック・メカニズムの一部であることを特徴とする方法。

【請求項２１】　請求項１８に記載の方法において、時間インターバル情報はデータ・サンプルに関するタイムスタンプ情報によって得られることを特徴とする方法。

【請求項２２】　請求項１８に記載の方法において、時間インターバル情報はデータ・ストリーム規則によって得られることを特徴とする方法。

【請求項２３】　複数のメディア・ストリームを同期化する方法であって、

メディア・サンプルの参照メディア・ストリームに基づいてマスタ・クロック参照を作成するステップであって、各サンプルは参照メディア・ストリーム内の位置を示す時間インターバル情報をもち、マスタ・クロック参照はタイムスタンプ情報から作成されるステップと、

１つまたは２つ以上の他のメディア・ストリームをマスタ・クロック参照に基づいて処理し、他のメディア・ストリームが参照メディア・ストリームの処理と同期化されるようにするステップとを備えることを特徴とする方法。

【請求項２４】　請求項２３に記載の方法において、マスタ・クロック参照はプロパティ集合の一部としてアクセスされることを特徴とする方法。

5

【請求項２５】　請求項２３に記載の方法において、マスタ・クロック参照は、サード・パーティ・エージェントからの要求に応答して作成される、選択的にインスタンス生成可能なクロック・メカニズムの一部であることを特徴とする方法。

【請求項２６】　処理コンポーネントを通して処理されるメディア・ストリームをハードウェア・レンダラに合わせてレート整合する方法であって、

ハードウェア・レンダラのハードウェア・オシレーションに基づく物理時間参照を処理コンポーネントに提供するステップと、

処理コンポーネントで行われるメディア・ストリームの処理レートを、物理時間参照に基づいてハードウェア・レンダラの処理レートに整合するように調整するステップであって、メディア・ストリームは時間インターバル情報が関連づけられているメディア・サンプルから構成されているステップとを備えることを特徴とする方法。

【請求項２７】　請求項２６に記載の方法において、物理時間参照はプロパティ集合の一部としてアクセスされることを特徴とする方法。

【請求項２８】　請求項２６に記載の方法において、物理時間参照は、サード・パーティ・エージェントからの要求に応答して作成される、選択的にインスタンス生成可能なクロック・メカニズムの一部であることを特徴とする方法。

【請求項２９】　請求項２６に記載の方法において、時間インターバル情報はメディア・サンプルに関するタイムスタンプ情報によって得られることを特徴とする方法。

【請求項３０】　請求項２６に記載の方法において、時間インターバル情報はメディア・ストリーム規則によって得られることを特徴とする方法。

【請求項３１】　第１コンポーネントにおける位置時間値を第２コンポーネントにおける位置時間値に基づいて、および共通ハードウェア・オシレータを利用して変換する方法であって、

相関時間値をシングル・オペレーションで第２コンポーネントから第１コンポーネントで受信するステップであって、相関時間値は時間インターバル情報が関連づけられているデータ・ストリーム内の位置に基づく位置時間値と、共通ハードウェア・オシレータに基づく共通時間値とを含んでいるステップと、

共通ハードウェア・オシレータの現在値を第１コンポーネントによって照会するステップと、

第１コンポーネントにおける位置時間値を共通ハードウェア・オシレータの現在値と第２コンポーネントから受信した相関時間値に基づいて第１コンポーネントによって変換するステップとを備えることを特徴とする方法。

【請求項３２】　第１コンポーネントにおける指定時間値を第２コンポーネントにおける指定時間値に基づいて、および共通ハードウェア・オシレータを利用して変換する方法であって、

6

相関時間値をシングル・オペレーションで第２コンポーネントから第１コンポーネントで受信するステップであって、相関時間値は指定時間値と共通ハードウェア・オシレータに基づく共通時間値とを含んでいるステップと、

共通ハードウェア・オシレータの現在値を第１コンポーネントによって照会するステップと、

第１コンポーネントにおける指定時間値を共通ハードウェア・オシレータの現在値と第２コンポーネントから受信した相関時間値に基づいて第１コンポーネントによって変換するステップとを備えることを特徴とする方法。

【請求項３３】　請求項３２に記載の方法において、指定時間値は時間インターバル情報が関連づけられているデータ・ストリーム内の位置に基づく位置時間値であることを特徴とする方法。

【請求項３４】　請求項３２に記載の方法において指定時間値はハードウェア・オシレータに基づく物理時間値であることを特徴とする方法。

【請求項３５】　２つのデータストリームの同期処理を可能にするカーネル・モード・データ処理システムにおいて、

第１データ・ソースと、

発生コンポーネントと終結コンポーネントを含む複数の第１カーネル・モード・データ処理コンポーネントであって、発生コンポーネントは第１データ・ソースから送られてきた第１データ・ストリームのデータ・サンプルを読み取り、処理コンポーネントの少なくとも１つはマスタ・クロック・コンポーネントが関連づけられているものと、

第２データ・ソースと、

発生コンポーネントと終結コンポーネントを含む複数の第２カーネル・モード・データ処理コンポーネントであって、発生コンポーネントは第２データ・ソースから送られてきた第２データ・ストリームのデータ・サンプルを読み取り、処理コンポーネントの少なくとも１つは複数の第１処理コンポーネントに置かれている、処理コンポーネントに関連づけられているマスタ・クロック・コンポーネントに基づいて処理を同期化するものと、

複数の第１と第２のそれぞれのカーネル・モード・データ処理コンポーネントのデータ処理コンポーネント間のカーネル・モード・コンポーネント相互接続であって、それぞれのデータ・サンプルの処理をそれぞれの発生コンポーネントから発生コンポーネントへ転送するようにしたものとを備えていることを特徴とするカーネル・モード・データ処理システム。

【請求項３６】　２つのメディア・ストリームの同期処理を可能にするカーネル・モード・メディア・レンダリング・システムにおいて、

第１メディア・ソースと、
発生コンポーネントと終結コンポーネントを含む複数の第１カーネル・モード・メディア処理コンポーネントであって、
発生コンポーネントは第１メディア・ソースからの第１メディア・ストリームのメディア・サンプルを読み取り、
終結コンポーネントは前記第１メディア・ストリームをレンダリングし、
各メディア処理コンポーネントはメディア処理コンポーネント間で第１メディア・サンプルを受け渡しするための接続ピン・インスタンスをもち、
少なくとも１つの接続ピン・インスタンスはマスタ・クロック・コンポーネントが関連づけられているものと、
第２メディア・ソースと、
発生コンポーネントと終結コンポーネントを含む複数の第２カーネル・モード・メディア処理コンポーネントであって、
発生コンポーネントは第２メディア・ソースからの第２メディア・ストリームのメディア・サンプルを読み取り、
終結コンポーネントは前記第２メディア・ストリームをレンダリングし、
各メディア処理コンポーネントはメディア処理コンポーネント間で第２メディア・サンプルを受け渡しするための接続ピン・インスタンスをもち、
少なくとも１つの接続ピン・インスタンスは、複数の第１処理コンポーネントに置かれている処理コンポーネントのピン・インスタンスの１つと関連づけられているマスタ・クロック・コンポーネントを使用して第２メディア・ストリームの処理を第１メディア・ストリームの処理と同期化するものと、
複数の第１と第２のそれぞれのカーネル・モード処理コンポーネントに置かれていて、それぞれの接続ピン・インスタンスを使用して作成されたそれぞれのメディア処理コンポーネント間のカーネル・モード・コンポーネント相互接続であって、それぞれのメディア・サンプルの処理コントロールをそれぞれの発生コンポーネントからそれぞれの終結コンポーネントへ転送するようにしたものと
を備えていることを特徴とするカーネル・モード・メディア・レンダリング・システム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明の分野は、発生クロックとレンダリング・クロックとの間のストリーム同期化およびレート整合といったように、マルチメディア・アプリケーションで起こるタイミング問題に関係する方法とコンピュータ・プログラム・プログラムに属するものである。また、本発明はソフトウェア・ドライバにおける標準化タイミング・メカニズムに関する。より具体的には、本発明は相互接続されたソフトウェア・ドライバなどの、複数のエンティティによって使用されるタイミングおよびクロック・メカニズムを標準化された方法で提供し、複数の処理されるデータ・ストリームが同期化され、異なるハードウェア・クロック間のレートが整合されるようにする方法およびコンピュータ・プログラム・プロダクトに関する。

【０００２】

【現在の技術状況】ディジタル化サウンドやビデオ・データなどのマルチメディア・データを処理するとき、連続するメディア・サンプルのストリームを処理しているのが普通である。また、データ・ストリームは、規則またはタイムスタンプ情報のどちらかによってデータと関連づけられた時間インターバル情報ももっている。タイミング情報は、特定のサンプルをいつ処理すべきかを処理コンポーネントに知らせるものであるが、処理がタイムスタンプ間で特定のレートで進行するのを保証するためにトラッキング参照(tracking reference)としても使用されている。

【０００３】処理が行われている期間に使用される時間インターバル情報を伝達するものとして、データ編成(data organization)、フォーマット、および規定(convention)もある。例えば、３０フレーム／秒（ｆｐｓ）フォーマットと規定のビデオ・データ・ファイルは、処理コンポーネントがその規定とフォーマットを知っていれば、各サンプルが１／３０秒のインターバルで処理されるものと認識される。

【０００４】メディア・ストリームを処理するときに起こる１つの問題は、２つまたはそれ以上のストリームを同期化してこれらが一緒にレンダリングされるようにすることである。例えば、サウンドトラックを表すオーディオ・ストリームと、それに付随するビデオ・データ・ストリームはコヒーレントなマルチメディア・プレゼンテーションが得られるように同期化する必要がある。

【０００５】ある同期化方法では、ＰＣクロックのような物理クロックやハードウェア・クロックに基づく値をもつクロック・メカニズムを利用している。データ・ストリーム・プロセッサはこのクロック・メカニズムの値を照会してストリームの処理またはレンダリングを同期化することが可能になっている。同期化はハードウェア・クロックを基準にして行われるが、２つの異なるストリームは処理するイベントや遅延が各ストリームに対して独立に行われることがあるので、常に同期化されるとは限らない。

【０００６】別の同期化方法では、ストリームが処理されている間、「マスタ」・データ・ストリーム上のプレゼンテーション・タイムスタンプ値に基づく時間値を提供するクロック・メカニズムを利用している。マスタ・ストリームが休止または停止されると、時間値はＰＣク

ロックのような物理クロックまたはハードウェア・クロックを基準にするように切り替わる。第２データまたはメディア・ストリームのプロセッサはこのクロック・メカニズムの値を照会(query) できるが、別の機能が用意されているので、クロック・メカニズムは物理クロック上の特定の時間に、あるいは、物理クロックに基づく一定のインターバル（例えば、１０ｍｓごと）に、プロセッサに通知するようになっている。

【０００７】イベント通知は物理クロックに基づいているので、処理がマスタ・データまたはメディア・ストリームで休止または停止されたときでも、このようなイベントは引き続き起こる場合がある。通知は一時的に禁止することができるが、そうすると、余分の処理オーバヘッドが発生するという犠牲が伴い、処理コードがさらに複雑化することになる。通知が明示的に禁止されたときでも、通知が誤って行われるというタイミング上の問題が生じることがある。

【０００８】メディアまたはデータ・ストリームを処理するとき起こるもう１つの問題は、メディア・ストリームに関連する異なるハードウェア・クロックのレートを整合するという問題である。あるハードウェア・クロックはメディア・ストリームのメディア・サンプルを生成し、タイムスタンプ情報を各サンプルと関連づけるときに使用されている。他方、全く別のハードウェア・クロックはメディア・ストリームをレンダリングするために使用されている。２つのハードウェア・クロックは見かけ上同じレートで動作しているように見えるが、物理クロック・レートが変動すると、処理に影響することになる。

【０００９】ここでレート整合(rate matching) とは、物理クロックに起こる実際の周波数変動を補償する概念である。例えば、ライブ・オーディオのストリームがローカル・オーディオ・ハードウェア側でレンダリングするためにネットワーク接続から受信されるとき、ストリームの受信時のレートはローカル・オーディオ・ハードウェアが実際にストリームをレンダリングするときのレートに一致していない場合がある。このようなことが起こるのは、オーディオ・ハードウェア側のクリスタルが、ネットワーク接続の他端に置かれたオーディオ・ハードウェア側のクリスタルとは無関係であるためである。

【００１０】場合によっては、ローカル・オーディオ・ハードウェア側で微調整を行うようにすると、ローカル側をソース側が発生している周波数に近づけることが可能な場合もある。しかし、すべてのオーディオ・ハードウェアがこれを可能にしているとは限らない。ハードウェアによっては、周波数解像度にある種の制約があるため調整の精度が制限されているものや、許容される調整量が制限されているものがある。

【００１１】処理時の「ドリフト(drift) 」はこのレート差が原因で起こるため、ドリフト量を正確かつ迅速に判断するための方法が追求されている。さらに、ドリフトの補償は、大きな調整を低頻度で行なうよりも、小刻みに高頻度で行なうと最適化されるので、ドリフト補償がエンド・ユーザに気づかれることがなくなる。この気づかれやすい調整は、サウンド・データを「スキップする(skipping)」および／またはビデオ・データが「滑らかに動かなくなる(jerking) 」という形で現れる。

【００１２】メディアまたはデータ・ストリームを処理するときに起こるさらに別の問題は、ある時間フレームを別の時間フレームに変換する問題である。この問題は、「スレーブ」・ストリームのプレゼンテーション時間がマスタ・フレームの時間フレームに変換されるときの同期の間によく起こっている。エラー発生を最小限にしてこの変換を行なうための方法が探究されている。

【００１３】マルチメディア環境では、ソフトウェア・ドライバを相互接続し、実行優先度が非常に高く、セキュリティ保護度が低いオペレーティング・システム・モードで実行され、多くの場合、ドライバが直接に操作する実際のハードウェアにアクセスできるソフトウェア・ドライバで処理を行なうようにすると好都合である。多くのアプリケーションは、本明細書においてWindows NT用語で「カーネル・モード(kernel mode) 」と呼ばれる、この制約がゆるやかでパフォーマンス指向のモードで実行させると有利である。他の強固なオペレーティング・システムも機能的に同等のモードをもっている。このような相互接続ソフトウェア・ドライバの場合も、同じように複数のメディア・ストリームを同期化し、物理クロックのレートを整合する必要がある。

【００１４】

【発明が解決しようとする課題】現在、本明細書で用いられているカーネル・モード・ドライバを容易に使用することができないプログラムの主要例の１つとして、ユーザがフィルタと呼ばれる、異なる処理ブロックを選択し、１つに接続してマルチメディア・データ・ストリームを連続的に操作できるようにするグラフ作成機能がある。データは、サウンドまたはビデオを表す一連のサンプルであるのが代表的であり、処理ブロックとしては、圧縮データのデコンプレション(decompression) 処理、特殊効果処理、CODEC 機能、データをアナログ信号にレンダリングすることなどがある。

【００１５】このようなフィルタはユーザ・モードで置かれているのが代表的であるので、プログラムのグラフ作成部分は、フィルタのオペレーションを相互接続してその制御を行い、ユーザの入力と処理ブロックの配置替えに応答できるようになっている。マルチメディア・データ・ストリームはその性質上一貫性があり、大量のデータが生成されるので、パフォーマンスは重大な問題となっている。汎用オペレーティング・システムでは、ユーザ・モードとカーネル・モードとの間の受渡し／切替

えを繰返し行うと、システム・パフォーマンスはある種のマルチメディア・アプリケーションが禁止されるほどに低下することがある。

【００１６】カーネル・モード・フィルタの相互接続を標準化する方法が望まれているが、標準化クロック・メカニズムがあると、複数のマルチメディア・ストリームを同時に処理するとき起こる同期化の問題が解決しやすくなるという利点がある。さらに、このような標準化クロック・メカニズムはデータを送信し、異なる物理クロックを使用してデータをレンダリングするとき起こるレート整合問題も取り扱って、解決する必要がある。

【００１７】本発明の目的は、各々が、関連する時間インターバル情報を有するデータ・サンプルからなる複数のデータ・ストリームの処理を同期化することである。

【００１８】本発明の別の目的は、データ・ストリームがストリーム・データ・サンプルに表された発生側物理クロックと、ハードウェアの一部と関連づけられたレンダリング側物理クロックとによって発生された場合にデータ・ストリームの処理レートの整合を容易化することである。

【００１９】さらに、本発明の目的は、相互接続されたカーネル・モード・ソフトウェア・ドライバのシステムで使用される標準化インタフェースをもつ有用なクロック・メカニズムを提供することである。

【００２０】さらに、本発明の別の目的は、一方のクロック・メカニズム上の時間から他方のクロック・メカニズム情報の時間への変換を容易化することである。

【００２１】本発明のその他の目的および利点は以下の説明に記載されているが、その一部はその説明から理解されるものと、本発明を実施することにより知得し得るものとがある。本発明の目的と利点は、請求の範囲に個々に記載されているインストリューメント(instrument)の手段および組合せにより実現し、得られるものである。

【００２２】

【課題を解決するための手段】上記目的を達成するために、また本明細書に具現化され、広い意味で説明されている本発明によれば、標準化クロック・メカニズムを使用して複数のデータ・ストリームを同期化し、異なる処理レートを整合する方法とコンピュータ・プログラム・プロダクトが提供されている。クロック・メカニズムは種々の異なる方法で実現することが可能であるが、本明細書では、データ・ストリームの処理を効率化する相互接続可能なカーネル・モード・ドライバを中心にして説明されている。

【００２３】本発明のクロック・メカニズムは３つの異なる時間値をもち、他のコンポーネントが利用できるようにする。これらの時間値を照会することも、クロック・メカニズムが一定の時間値に来たことに基づいて、または周期的インターバルで通知を送ることも可能である。

【００２４】本発明のクロック・メカニズムで利用できる最初の時間値は位置時間値(positional time value)であり、これはデータ・ストリームに関連する時間インターバル情報に基づいており、処理されるデータ・ストリーム内の位置を反映している。第２の時間値はＰＣクロックやレンダリング・プロセッサ・クロックなどの実際のハードウェア・オシレータまたはクロックに基づく物理時間値である。最後に、第３の時間値は相関時間値(corrclated time value)であり、これは位置時間値などの指定時間値(designated time value)をＰＣクロックなどの共通に利用可能なハードウェア・クロックに基づく参照時間値と共に提供する。

【００２５】クロック・メカニズムをマスタ・クロックとしたときは、位置時間値は他のデータ・ストリームを処理する他の処理コンポーネントが同期をとるために使用することができる。位置時間値はデータ・ストリームの処理に基づいているので、基礎となるデータ・ストリームの処理が停止すると、クロックも停止する。さらに、タイミング・イベント通知も絶対時間を基準にして凍結されるが、位置時間値に対して完全にタイミングは合い続ける。このことは、「休止された」時間をトラッキング（追跡）したり、操作したり、あるいは気にしたりしないで済むという大きな利点がある。「スレーブ」・クロック・メカニズムとコンポーネントは位置時間値から得られる時間を基準にして処理を行う。

【００２６】データ・ストリーム・レンダラまたは他のプロセッサに関連する基礎となるハードウェア・クロックの物理時間値は、処理されるデータ・ストリームのレートをレンダリング・プロセッサの処理レートと一致させるために使用できる。データ・ストリームは関連づけられている時間インターバル情報を有しているので、発生側プロセッサの処理レートは固有であるので、これを確かめて、物理時間値に入っているレンダリング・プロセッサの実際の処理レートと比較することができる。これにより、処理レートの「ドリフト」を迅速かつ正確に判断できるので、レンダリングされたデータに目立った影響を与えることなく処理の調整を行うことができる。

【００２７】最後に、相関時間値は他のクロック・メカニズムと処理コンポーネントが時間情報をある基準から別の基準へ変換するときの手段となるものである。共通ハードウェア・クロックは共通の参照として使用できるように、他のクロック・メカニズムまたはコンポーネントが利用できるようになっていなければならない。相関時間はアトミック・オペレーションとして返されるまたはアクセスされるので、発生する誤差は最小限であり、非累積的になっている。

【００２８】相関時間はここで説明している実施の形態では２種類がサポートされ、どちらの場合も、ＰＣクロックを参照として使用している。一方はメディア時間を

指定時間(designated time) として使用し、他方は物理時間を指定時間として使用している。本明細書では、相関時間が単独で使用されるときは、２種類の相関時間のどちらか、またはこの分野の当業者ならば自明である他の種類の相関時間を意味している。これらを区別する必要があるときは、相関メディア時間と相関物理時間の用語が使用され、それぞれの指定時間が共通参照と比較されるようになっている。

【００２９】本発明の一実施の形態は、標準化された方法でカーネル・モード・ドライバを接続する一部として行われる。ある所定のドライバまたはフィルタは接続ポイントの「ピン・ファクトリ(pin factory) 」をサポートし、定義している。ピン・ファクトリは他のドライバ上の他のピン・インスタントと相互接続される接続ピン・インスタンスをインスタンス生成するために使用され、ユーザ・モード・エージェントに頼らなくても、処理メッセージをドライバがカーネル・モードで連続的に処理できるようにする。

【００３０】コンプライアント・ドライバ(compliant driver)を接続することを希望するサード・パーティ・エージェントはドライバに能力(cabability)があるかどうかを照会する。そのような能力としては、どの種類の接続ピン・インスタンスがインスタンス生成できるか、といったことがあり、この中には、処理されるデータのタイプ、データ・フォーマット、転送レート、転送媒体またはモード、接続ピン・インスタンスの入力または出力の性質などの関係特性が含まれる。照会されるものとして、ほかにも、必要なデータ・バッファ容量と、各接続ピン・ファクトリで利用できるバッファを割り当てる能力がある。

【００３１】ユーザ・モードで実行されるのが代表的であるサード・パーティ・エージェントが１つまたは２つ以上のコンプライアント・ドライバの機能について照会したあと、そのエージェントは、複数のドライバを１つに「チェイニング」してデータがドライバ間で最適に処理されるようにする最良の接続特性を決定する。この決定ステップは、すべてのドライバ機能の照会を終えて、最適な接続基準が選択できるようになったあとで行われる。

【００３２】各接続ピン・インスタンスごとに、サード・パーティ・エージェントはバッファ・アロケータ(buffer allocator)を特定のピン・インスタンスで作成する必要があるかどうかも判断する。この場合も、これはすべての異なるフィルタとピン・ファクトリの照会を終えたあとで行われ、そのあとで相互接続が行われる。

【００３３】そのあと、サード・パーティ・エージェントは、各ドライバ上の接続ピン・ファクトリに基づいて接続ピン・インスタンスを作成することによってドライバを相互接続する。エージェントは接続ピン・インスタンス作成の一部としてデータ・フォーマットと接続フォーマットを指定する。ＮＴオペレーティング・システムの下で実装された実施の形態では、実際の接続ピン・インスタンスは「ファイル」へのハンドルを戻す作成入出力オペレーションによって作成される。作成入出力要求はドライバ・インスタンス・ハンドルと、接続ピン・インスタンスのデータ・フォーマットと接続フォーマットを示しているデータ構造への参照とを収めている。

【００３４】さらに、以前に作成された接続ピン・インスタンス（例えば、入力ピンまたはＩＲＰ「シンク」・ピン）への参照は、他の接続ピン・インスタンス（例えば、出力ピンまたはＩＲＰ「ソース」・ピン）を作成する要求の中で指定され、接続ピン・インスタンス間の接続が行われるようにする。ソース・ピン・インスタンスがバッファ・アロケータをもつ入力ピン・インスタンスへの参照を使用して作成されると、バッファ・アロケータのソース・ピン・インスタンスと一緒に通知が行われ、データが既存バッファから新しいバッファに送り込まれるようにする。参照が示されていなければ、ソース・ピン・インスタンスは処理を終えた後、データを既存バッファに残しておく。

【００３５】コンプライアント・ドライバを作成するためには、ドライバ開発者は、ユーザ・モードのエージェントが機能について照会し、ドライバ間の相互接続を行えるようにする、ある種の標準機能をサポートしている。Windows NTオペレーティング・システム上で構築されている、ある実施の形態では、これは、必要とする機能を実装している「集合(set) 」（つまり、プロパティ、メソッド、およびイベントの集合）の使用によって達成されている。

【００３６】集合は、集合全体を特定するＧＵＩＤ(globally unique identifier)（グローバルにユニークな識別子）と、集合内の各機能エレメントのＲＵＩＤ(relatively unique identifier)（相対的にユニークな識別子、例えば、集合自体内で相対的な識別子）をもつものとして論理的に定義されている。各集合は１つまたは２つのＩＯＣＴＬｓ(IO Controls)(入出力コントロール)だけと関連づけられ、ＩＯＣＴＬは、集合の仕様と組み合わされて、ドライバとのすべてのインタラクションを制御する。

【００３７】本実施の形態では、３タイプの集合が使用されている。つまり、プロパティ集合、メソッド集合、およびイベント集合である。プロパティ集合はサウンド・ボリューム、転送レートなどの、値または設定値をドライバ内で管理するために使用され、コールがプロパティ値を得ようとしているのか、および／またはプロパティ値を設定しようしているのかを示すフラグと一緒に単一のＩＯＣＴＬを使用する。メソッド集合は、メモリの割当て、バッファのフラッシング(flushing)などのように、ドライバが実行できるオペレーションを管理するために使用され、単一のＩＯＣＴＬを使用して特定のメソ

ッドをコールする。イベント集合は、デバイス変更通知、データ・スターベーション(data starvation) 通知などのように、ドライバの処理に関連するイベントを管理するために使用され、２つのＩＯＣＴＬを使用する。１つは特定のイベントをイネーブル(enable)にするためのもので、もう１つは特定のイベントをディスエイブル(disable) にするためのものである。

【００３８】集合を使用するには、入出力制御オペレーションは特定のＩＯＣＴＬと、集合のＧＵＩＤ、ＲＵＩＤをもつデータ構造への参照およびその他の必要なデータを使用して開始される。例えば、サウンド・カード・ドライバでボリューム・プロパティを設定する場合は、集合プロパティのＩＯＣＴＬを使用し、ボリューム設定値をもつプロパティ集合の適切なＧＵＩＤを指定し、その集合内の特定ＲＵＩＤがボリューム・プロパティを示していることを指示し、新しいボリューム設定値を含んでいる入出力制御オペレーションが行われることになる。

【００３９】サポートされる集合について照会するには、ヌルＧＵＩＤが特定集合タイプの特定ＩＯＣＴＬ（例えば、プロパティ集合ＩＯＣＴＬ、メソッド集合ＩＯＣＴＬ、またはイベント・イネーブルＩＯＣＴＬ）で照会フラグと一緒に使用され、サポートされる集合ＧＵＩＤのリストが返される。ある集合内のサポートされたプロパティ、メソッド、またはイベントについて照会するには、集合ＧＵＩＤ、集合タイプＩＯＣＴＬ、および照会フラグが、サポートされたＲＵＩＤのリストを返すオペレーションと一緒に使用される。

【００４０】汎用集合メカニズムを使用すると、コンプライアント・ドライバをサポートするために最小限の機能を実装することができるが、それでも拡張性は無制限である。集合は、特定の集合が実装されている限り、相互動作可能で、相互接続可能なドライバのシステムを作成するために、多数の異なるドライバ開発者によって独立してコーディングすることができる仕様書に定義することができる。さらに、この仕様書には、サポートする必要のある必須のプロパティ、メソッドおよびイベントを定義できるだけでなく、ドライバの機能と拡張機能に依存して実装できるオプションのプロパティ、メソッド、およびイベントも定義することができる。要求される基本的最小限の共通性のほかに、ドライバ開発者は独自の集合を定義し、その集合にＧＵＩＤを割当てることによって追加の機能を組み込むことも可能である。サポートされる機能（つまり、サポートされるＧＵＩＤとＲＵＩＤの照会を行うこと）を列挙できるようにすると、サード・パーティ制御エージェントなどの、呼出し側(caller)は期待を調整することも、基礎となるフィルタの機能に依存して適切な補償を行うこともできる。

【００４１】位置時間値、物理時間値、相関位置時間値、および相関物理時間値をもつクロック・メカニズムの実施の形態は、コンプライアント接続ピン・インスタンス上に作成することができる。「ファイル」は、接続ピン・インスタンスを親として表しているファイル・オブジェクトのハンドルと、クロック・メカニズムのファイル・タイプを示すＧＵＩＤストリング値を指定して、入出力の特定のデバイス上でオープンされる。ハンドルはクロック・メカニズムを表すファイル・オブジェクトに返され、フィルタはそれぞれの時間値を提供するプロパティ集合を、このファイル・オブジェクトを通してサポートする。位置時間値はフィルタによって処理され、特定の接続ピン・インスタンスに渡されるデータ・ストリームに基づいている。物理時間は、ハードウェア・クロックまたは存在すればフィルタに関連するオシレータ、つまり、ＰＣクロックに基づいている。相関時間値はどちらも、ＰＣクロックが普及しているので、ＰＣクロックを参照として使用しているのが代表的である。

【００４２】クロック・メカニズムを「マスタ」として使用すると、他のクロック・メカニズムまたはコンポーネントを「スレーブ」にし、あるいはその時間トラッキングと同期させることができる。さらに、ＰＣクロックに基づくデフォルトの実装が用意されている。

【００４３】本発明の上記およびその他の目的と特徴は以下の説明および請求の範囲に詳しく記載されている通りであるが、以下に説明する本発明を実施することによって知得することも可能である。

【００４４】本発明の上記およびその他の利点と目的がどのようにして得られるかという方法を示すために、以下では、添付図面に図示されている具体的実施の形態を参照して、以上で簡単に説明してきた本発明を詳しく説明することにする。添付図面は本発明の代表的な実施の形態を示したにずぎず、従って、本発明の範囲を限定するものではないとの了解の下で、以下、添付図面を使用しながら本発明をもっと具体的にかつ詳しく記述し説明することにする。

【００４５】

【発明の実施の形態】本明細書で用いられている「ユーザ・モード(user mode) 」という用語は、ユーザが書いたプログラムの大部分が実行されるときの、オペレーティング・システムにおける動作レベルのことである。ユーザ・モードの動作レベルはセキュリティ・レベルが最も高く、あるアプリケーション・プログラムまたはプロセスが別のアプリケーション・プログラムまたはプロセスに干渉するのを防止するために大量のオーバヘッドが消費されるのが代表的である。さらに、システム・リソースへのアクセスは特定のインタフェースを通して厳格に制御され、実行優先度は最低ではないとしても、最低優先度の１つであるのが一般である。

【００４６】本明細書で用いられている「カーネル・モード(kernel mode) 」という用語はユーザ・モードの動作モードよりも制約が非常に少ない、オペレーティング

・システムにおける動作レベルのことである。カーネル・モードのプログラムまたはプロセスの例としては、ハードウェア・コンポーネントを制御するソフトウェア・ドライバ(software driver) を含む。代表例として、カーネル・モードのプログラムはパフォーマンスに影響されやすく、従って、動作上のオーバヘッドはユーザ・モードのプログラムよりも少なくなっている。さらに、ハードウェアや多くのシステム・リソースへのアクセスは無制約であるか、あるいはユーザ・モードのプログラムの場合よりも制約がはるかに少なくなっている。多くの場合、カーネル・モードで実行されるプログラム・コードは、規則(convention)に対するプログラマの規律正しさと準拠とを信頼して、システムが良好に動作するようにする（例えば、別のプログラムのアドレス空間を破壊しないようにする）。カーネル・モードを表わす別の用語として、「トラステッド(trusted) 」コードがある。

【００４７】本明細書で用いられている「ドライバ(driver)」という用語はカーネル・モードで実行されるのが代表的である、ソフトウェア・ドライバ・プログラムのことである。ドライバという用語は、オペレーティング・システム上にロードされる実際の実行可能プログラムまたはある種の機能を分け与えるプログラム部分を意味する場合もある。ドライバは、多くの場合、ある形態のハードウェアと関連づけられているが、必ずしもそうである必要はない。

【００４８】本明細書で用いられている「フィルタ(filter)」という用語はソフトウェア・ドライバ内に置かれている機能部分のことであり、この中には、ドライバ自体も含まれ、そこでは、接続ポイントがフィルタを通してデータを送信するために公開されている。例えば、ソフトウェア・ドライバはいくつかの異なるフィルタをサポートする場合もあれば、１つの単一機能をもっている場合もある。さらに、内部的には１つに接続され、外部的には入出力用に接続ポイントを公開している異なるドライバからの複数のフィルタは単一フィルタとして集合的に参照される場合もある。また、もっと総称的な意味では、フィルタという用語は、伸張(decompression) などのように、それがカーネル・モードで実行されるソフトウェア・ドライバ・フィルタで起こるか、ユーザ・モードで実行される別のプログラム・コードで起こるかに関係なく、実行されるオペレーションを意味する場合もある。

【００４９】本明細書で用いられている「ドライバ・オブジェクト(driver object) 」という用語は、ソフトウェア・ドライバを管理し、これをシステム・リソースとして知らせるオペレーティング・システムによって定義されているオペレーティング・システムのエンティティのことである。

【００５０】本明細書で用いられている「デバイス・オブジェクト(device object) 」という用語はシステムによって定義されたシステム・レベルのエンティティのことであり、これは利用が可能であるドライバの機能の一部をシステム・リソースとして知らせ、ドライバの機能と他のシステム・コンポーネントが利用できるかどうかを定義している。ドライバ・オブジェクトとデバイス・オブジェクトはどちらも、代表的にドライバ・ソフトウェアのロード時と初期化時に作成される。

【００５１】本明細書で用いられている「ファイル・オブジェクト(file object) 」という用語は、デバイス・オブジェクトによって指定されたリソースの呼出しを管理するオペレーティング・システムのエンティティのことであり、これはオペレーティング・システムによって定義されている。ファイル・オブジェクトはドライバ・オブジェクトの使用状況に関するコンテキストを提供する。さらに、ファイル・オブジェクトは、以前のファイル・オブジェクトが新しいファイル・オブジェクトの作成時に「親」と指定されていれば、別のファイル・オブジェクトと階層的に関係づけることが可能である。ファイル・オブジェクトは、代表的にデータ・ストリーム上に操作するすべての入出力オペレーションの管理に使用される。

【００５２】本明細書で用いられている「データ(data)」という用語は、相互接続されたカーネル・モードのフィルタを通して処理される一切の情報のことである。そのようなデータは、ビデオ、オーディオ、テキスト、ＭＩＤＩなどを表すメディア・データを含むが、他のアプリケーションの場合には制御情報やパラメータも含んでいる場合がある。例えば、カーネル・モード・フィルタ・グラフはプロセス制御オペレーションで使用されることがあるが、そこでは、異なるフィルタ間で受け渡しされる制御情報はマシン類をアクチュエートする制御信号を創り出すために使用されている。メディア処理システムの例が示されているが、他のアプリケーションも、本明細書に説明されている相互接続カーネル・モード・フィルタのシステムから、同じような方法で利点を得ることが可能である。

【００５３】本明細書では、本発明の説明は、Microsoft(登録商標）社から提供されているWindows NT（商標）オペレーティング・システムのコンテキスト内で記載されている。さらに、本明細書に説明されている好適実施の形態を理解するためには、Windows NTの入出力アーキテクチャの知識が前提になっている。入出力システムおよび NT オペレーティング・システム全般について解説した好適書としては、Helen Custer著「Windows NTの内部(Inside Windows NT) 」(Microsoft Press発行）があるが、本書は引用により本明細書の一部を構成するものである。

【００５４】ドライバおよびファイル・オブジェクト、デバイス・オブジェクト、ドライバ・オブジェクトなどのシステム・エンティティの以下の説明では、Windows

NTオペレーティング・システム上でこれらがどのような働き方をするかが解説されているが、この分野の精通者ならば理解されるように、本発明は、類似のコンポーネントを持つ他のオペレーティング・システム上で実装することが可能であり、これらのオペレーティング・システムが同じ用語を使用しているかどうかとは無関係である。例えば、別のオペレーティング・システムがファイル・オブジェクトとして動作するエンティティを持っていれば、そのエンティティはその実際の名称に関係なく、ファイル・オブジェクトと解釈することができる。

【００５５】まず、図１を参照して説明すると、図示のシステム例は、サウンド・データのストリームをディスク・ドライブから読み取り、そのサウンド・データをレンダリング(rendering) して、従来のモデルに従ってスピーカから聞こえるようにするものである。あるデータ量はハード・ドライブ２０にストアされるが、これはディジタル化されたサウンド・サンプルの形態でサウンドを表している。ほかにも、サウンド・データ・ストリームのソースとしては、電話回線を利用して送られてくるディジタル化情報、ネットワークや他の通信パケットからのディジタル化情報、この分野で公知の他のソースが考えられる。データ・ストリームはディジタル化サンプルから構成され、これらのサンプルはデータ・フォーマットと規則によって、または各サンプルに付加される明示的タイムスタンプ情報によってタイム・インターバル情報が関連づけられている。カーネル・モードのディスク・ドライバ２２はディスク・ドライブ・ハードウェア２０と相互作用し、ユーザ・モードのリーダ(reader)プログラム・コンポーネント２４の制御下に置かれている。制御エージェント(controlling agent) ２６はサウンド・データのレンダリングを行うために異なるコンポーネントを管理するが、動的グラフ作成機能(dynamic graph building capability) を備えている場合は、異なるソフトウェア・コンポーネントが動的に割当てられて、エンド・ユーザの指定に従ってカスタム・フィルタリングまたは他の処理経路を提供できるようにする。

【００５６】リーダ・コンポーネント２４はオペレーティング・システムの標準入出力制御インタフェースを使用してディスク・ドライバ２２と相互作用し、圧縮サウンド・データをディスク・ドライブ２０から読み取って、ユーザ・モード・プロセスのアドレス空間(address space) の一部としてユーザ・モードで割当てたバッファに入れる働きをする。次に、デコンプレッサ(decompressor)・コンポーネント２８は圧縮データを処理に適した伸張(decompressed)フォーマットに伸張する。図示のように、このステップ全体は、付随の低優先度の、プロセス動作(processbehavior)安全メカニズムを使用してユーザ・モードで実行される。

【００５７】効果フィルタ(effects filter)３０はデータに操作を加えて、ある種の特殊効果が得られるように、カーネル・モードで動作する附属の効果フィルタ３２をもっている。さらに、効果プロセッサ３４が存在する場合もあれば、効果フィルタ全体が実際のハードウェア・プロセッサをエミュレートするソフトウェアで動作する場合もある。効果フィルタ３２をアクセスするためには、効果コンポーネント３０はシステム入出力制御メカニズムを使用して、データと制御を効果フィルタに転送する。この場合も、カーネル・モード／ユーザ・モード境界を交差して、この遷移が行われる。

【００５８】効果フィルタ３２は効果プロセッサ３４を制御し、必要または望ましいとされる特殊効果がそのデータ上に作られるようにする。これは、効果コンポーネント３０から全データをコピーし、そのコピーを効果フィルタ３２に、実際のシステム構成によっては、再度効果プロセッサ３４にも移すことにより行われる。多くのソフトウェア効果コンポーネントはハードウェア・プロセッサが関連づけられているが、他のコンポーネントは、ホスト・プロセッサ上で実行されるシステム・ソフトウェア内で完全に機能している。

【００５９】効果コンポーネント３０の処理の完了時にコントロールとデータがユーザ・モードに戻されると、これは、次に、サウンド・レンダリング・コンポーネント３６に転送される。サウンド・レンダリング・コンポーネント３６はコントロールとデータをサウンド・レンダリング・ドライバ３８に転送し、これを受けてサウンド・レンダリング・ドライバは、処理され、フィルタリングされたデータをスピーカ４２からのサウンドとしてレンダリングするようにサウンド・カード４０を制御する。以上から容易に理解されるように、ユーザ・モードとカーネル・モード間の転送は多種類存在するために、サウンド・データのレンダリングを非効率化している。連続するサウンドまたはビデオ・ストリームのように、マルチメディア・データはタイミングに影響されやすい性質のため、これらの非効率性とコントロールの遷移を少なくすると共に、異なるバッファ間でなん度もデータをコピーすることを少なくすると、好都合である。

【００６０】本明細書で用いられている本発明の一実施の形態は、Windows NTオペレーティング・システム・アーキテクチャ上で提供されるサービスから構成されている。このサービスはシステムのユーザがアクセスする、いくつかの異なるソフトウェア・コンポーネントに分割されている。第１は、ユーザ・モードのＡＰＩであり、これは、接続ピン・インスタンスと、クロック・メカニズムやバッファ割当てメカニズムなどの特定の機能を表している他のファイル・オブジェクトとを作成するためのルーチンを含んでいる。ほかにも、もっと重要なものとして、ドライバ開発者が標準化アーキテクチャに従うドライバを作成するのを支援するルーチンとデータ構造が完備されている。システムに用意されているこれらの機能を利用すると、異なるドライバ開発者は特定のアー

キテクチャに準拠して相互に相互作用するコンプライアント・ドライバを作成することができる。ユーザ・モード・エージェントはＮＴエグゼクティブおよび入出力マネージャのシステム・サービスと通信する、ユーザ・モードで実行中の環境サブシステムを通してコンプライアント・ドライバと通信する。これは、他のすべての入出力に対する同じ標準入出力メカニズムであり、好適実施の形態の本実装では、既存のシステム・サービスを可能な限り利用するようになっている。

【００６１】本発明を利用する図１のシステムのアーキテクチャは図２に示すようになっている。制御エージェント４４は知らされているドライバに照会し、データ・フォーマットと接続フォーマットに従って相互接続を行い、レンダリングを完全にカーネル・モードで行うようにする。さらに、制御エージェントは重要なイベントの通知を受けるので、必要に応じて制御を行うことができる。このようなイベントの例としては、処理の終了、データ・スターベーション事態などを含む。

【００６２】この構成では、サウンド・データは前述したように、ディスク・ドライバ４８によってディスク・ドライブ４６から読み取られる。リーダ・ドライバ５０はディスク・ドライバ４８を制御し、従来の使い方と同じようにＮＴ層(NT layered)入出力アーキテクチャに従ってディスク・ドライバ４８と「垂直方向」に関連づけられている。「垂直方向」と「水平方向」の用語は、ＮＴ層入出力アーキテクチャの一部として現在行われているドライバ接続（垂直方向）と、サード・パーティ制御エージェントによって動的に行われる相互接続カーネル・モード・ドライバに従う接続（水平方向）とを区別するために用いられている。

【００６３】リーダ・ドライバ５０は以下で説明する接続メソッドに従ってデコンプレッサ・ドライバ５２とも「水平方向」に相互接続され、制御エージェント４４によって管理されている。デコンプレッサ５２は伸張をカーネル・モードで行ってから、データとコントロールを効果フィルタ５４に引き渡す。効果フィルタは必要に応じて効果プロセッサ５６を利用して特殊効果を適用してからデータとコントロールをサウンド・レンダリング・ドライバ５８に引き渡し、このドライバはサウンド・カードを制御し、データをスピーカ６２からのサウンドとしてレンダリングする。図２から理解されるように、処理をカーネル・モードのままにしておくと、ユーザ・モードとカーネル・モードとの間の複数の遷移がなくなり、処理をユーザ・モードで行うと通常起こるオーバヘッド量が減少するので、効率面の利点が得られる。

【００６４】次に、図３を参照して説明する。図３は、本発明の一実施の形態に従う相互接続ソフトウェア・ドライバに関係するシステム・オブジェクトの階層的性質を示すロジック図である。ドライバ・オブジェクト６４は、メモリにロードされるときの、実行可能ソフトウェア・コード・イメージを表すために作成される。ドライバ・コード・イメージはドライバの機能全体を含んでおり、ドライバ・オブジェクト６４はシステムに置かれている場所、サポートされるドライバの種類などのイメージに関する情報を含んでいる。

【００６５】制御エージェントによって独立にアクセスできる機能の各タイプ別に、デバイス・オブジェクト６６$_a$～６６$_n$が入出力ディレクトリ構造に作成され、これらのオブジェクトは利用可能で、ユーザ・モードのクライアントによってアクセスされる異なる機能を表している。これらは、フィルタまたは独立に利用できる他の機能部分を表しているのが代表的である。ドライバ・オブジェクト６４とデバイス・オブジェクト６６$_a$～６６$_n$は囲みボックス６８で示すように、ドライバ・コードのインストール時に作成される。

【００６６】歴史的には、デバイス・オブジェクトは物理ハードウェアの各エレメントごとに存在する。しかるに、最新の入出力システムの柔軟性は、デバイス・オブジェクトが完全にソフトウェアで実装されたフィルタを表すことを可能にさせている。そのため、デバイス・オブジェクトは専らソフトウェアで実装されたフィルタの各インスタンスごとに作成することが容易になっている。従って、ソフトウェア・フィルタは、デバイス・オブジェクトで表された各インスタンスがデバイス・オブジェクトと１対１の対応関係をもつように実装することも、単一のデバイス・オブジェクトがより伝統的な手法に従い、各々がフィルタのクライアント・インスタンスを表している複数のファイル・オブジェクトを管理するように実装することも可能になっている。

【００６７】デバイス・オブジェクト、図示の例では、デバイス・オブジェクト６６$_a$上には、ファイル・オブジェクトが作成され、これはデバイス・オブジェクトで表された機能の独立インスタンスを表している。デバイス・オブジェクトはフィルタを表し、そのフィルタの複数のインスタンスを管理するのに対し、ファイル・オブジェクトは特定のエンティティによって使用される、そのフィルタの実際のインスタンスを表している。従って、ファイル・オブジェクト７０はデバイス・オブジェクト６６$_a$によって定義されたフィルタのインスタンスである。

【００６８】フィルタを使用するには、制御エージェントまたは他のユーザ・モード・クライアントは入出力ディレクトリ構造内で利用可能なデバイス上でファイルをオープンする。適切なコンテキスト情報を収めているファイル・オブジェクトが作成され、そのファイルへのハンドルがユーザ・モード・クライアントに返される。ファイル・オブジェクトは、作成時に「親」ファイル・オブジェクトを指定することによって階層的に関係づけることが可能であるが、ファイル・オブジェクトはすべてが同一デバイス・オブジェクトの子であるような、兄弟

(sibling) 関係も持っている。

【００６９】ファイル・オブジェクト内のコンテキスト情報は、ユーザ・モード・クライアントとの入出力インタフェースを管理する情報、ファイル・オブジェクトが表わしているエンティティの「状態(state) 」などからなっている。コンテキスト情報には、システムが必要とする情報があり、さらに、特殊な意味をもたせることができる、ユーザ定義可能エリアも含まれている。ユーザ定義可能エリアの使い方の例は、下述するバリデーション(validation)とＩＲＰルーチング(routing)・メソッドのインプリメーテンションの説明個所に示されている。

【００７０】接続ピン・インスタンスを提供するためには、フィルタ・インスタンスを表すファイル・オブジェクト７０が親として使用され、特定フィルタの接続ピン・インスタンスを表す子ファイル・オブジェクトが作成される。ファイル・オブジェクト７０は接続ピン・ファクトリの定義と可用性に関して照会されるのに対し、実際のファイル・オブジェクトは特定のファイル・オブジェクトを適切な情報コンテキストとして使用して、ピン・ファクトリの各インスタンスごとに作成され、接続ピン・インスタンスが有効に、かつ正しく作成されるようにする。例えば、ファイル・オブジェクト７２と７４はファイル・オブジェクト７０で表されたフィルタの接続ピン・インスタンスを表し、ファイル・オブジェクト７０と階層的に関係づけられている。それぞれファイル・オブジェクト７２と７４で表された接続ピン・インスタンスはフィルタ・インスタンス（ファイル・オブジェクト７０で表されている）に入ったあとで、そこから出るデータ・パスにすることができ、これは一連のチェイン・フィルタまたは他のドライバ機能を形成するときに他の接続ピン・インスタンスと接続するために使用できる。

【００７１】ピン・インスタンスがフィルタ・インスタンスを表す別のファイル・オブジェクトと階層関係をもつファイル・オブジェクトで表されて、ピン・インスタンスのコンテキスト情報を提供するようにするのとまったく同じように、他のファイル・オブジェクトをピン・インスタンスと階層的に関係づけて他の機能を表すようにすると、正しいコンテキスト情報が得られるようになる。コンテキスト情報は、ピン・データ・フォーマット、通信タイプなどのように、作成時に使用される個々のパラメータに従って、あるピン・インスタンスを他のピン・インスタンスと区別するために必要である。

【００７２】バッファ割当てメカニズム、タイミング・メカニズムなどのように、個別コンテキストかユーザ・モードのコントロールのどちらかをハンドルを通して要求する他の操作エンティティも、ファイル・オブジェクトで表すことができる。さらに、ファイル・オブジェクト（例えば、特定の接続ピン・インスタンスと関連づけられたバッファ割当てメカニズム）間の階層関係は、必要ならば、子ファイル・オブジェクトの作成時に親ファイル・オブジェクトを指定することにより確立することができる。これらの親子関係は、操作エンティティを表すファイル・オブジェクト間の関係と構造を決定するために存在する。さらに、特定タイプの「親」ファイル・オブジェクトはある種のタイプの「子」ファイル・オブジェクトだけを作ることができるので、以下で説明するように作成バリデーション・メカニズムが必要になる。この場合も、このようなファイル・オブジェクトは対応するハンドルがユーザ・モードで利用可能になっており、これらのハンドルはNtCreateFileなどのシステムＡＰＩコールを通してクライアントに返される。

【００７３】ファイル・オブジェクトへのハンドルはカーネル・モード・ドライバと通信するために、制御エージェントなどのユーザ・モード・クライアントによって使用される。ファイル・オブジェクト、デバイス・オブジェクト、およびドライバ・オブジェクトの階層的チェインは、入出力サブシステムが階層関係をもつファイル・オブジェクトとデバイス・オブジェクトを通ってドライバ・オブジェクトまで戻り、実際のドライバ・コードに入るエントリ・ポイント(entry point) に到達することができるようにする。このようなエントリ・ポイントはソフトウェア・ドライバ・コードの中の機能を指す参照（例えば、ポインタ）になっている。さらに、特定のファイル・オブジェクトと、ソフトウェア・ドライバ・コードへのエントリ・ポイントをもつドライバ・オブジェクトとの間のオブジェクト通路(pathway) 上にあるオブジェクトの各々は、入出力サブシステムがＩＲＰを作成するときに重要なコンテキスト情報のほかに、下述するルーチングおよびバリデーション・メカニズムに従ってＩＲＰを正しくルーチングするときに使用されるデータ構造への参照も提供する。

【００７４】ファイル・オブジェクトと他のシステム・オブジェクトに対するハンドルはプロセス専用であり、ユーザ・モード・プロセスが基礎となるオブジェクトと通信するときの手段となる。注目すべきことは、ファイル・オブジェクトなどの、基礎となる単一システム・オブジェクトを参照するために複数のハンドルが作成できることである。このことは、複数のアプリケーションがファイル・オブジェクトで表されたピン・インスタンスに情報を供給できることを意味する。

【００７５】異なるドライバを相互接続するとき重要となる情報エレメントの１つとして、デバイス・オブジェクト・スタックの深さ(depth) パラメータがある。これは特定のドライバ・オブジェクトのＩＲＰスタック・ロケーションを示している。このようにすると、入出力マネージャを使用して、相互接続されたドライバ間で単一のＩＲＰを使用し受け渡しできるので、ＩＲＰを別々に作成し、それを種々の相互接続ドライバ間で送信するよ

りもパフォーマンス向上を提供できることになる。別の方法として、各ドライバは適切な入出力マネージャのコールを通して、各連続通信ごとに新しいＩＲＰを作成し、各々の新ＩＲＰを相互接続されたドライバのチェイン上の次のドライバへ送信させることも可能である。

【００７６】次に、図４～図６を参照して説明する。図は、異なるタイプのファイル・オブジェクト作成のバリデーションと適切なハンドラへの入出力要求パケット（I/ORequest Packet：ＩＲＰ）のルーチングを可能にするシステム・ドライバ・オブジェクト、デバイス・オブジェクト、およびファイル・オブジェクトのエクステンションを示している。図４は、１つまたは２つ以上のフィルタまたは他のドライバ機能を実装している実行可能コードを表すドライバ・オブジェクト７６を示している。ドライバ・オブジェクト内では、Windows NTアーキテクチャはソフトウェア・ドライバ開発者が用意した作成ハンドラへの参照を要求している。この実施の形態によれば、マルチプレクシング・ディスパッチ機能(multiplexing dispatch function)７８は作成ハンドラとしてドライバ・オブジェクト７６から参照され、メッセージを作成されるファイル・オブジェクトのタイプに応じて特定の作成ハンドラへルーチングするために使用される。マルチプレクシング・ディスパッチ機能７８のオペレーションは図８に示すフローチャートを参照して以下で説明する。

【００７７】同じように、ドライバ・オブジェクトからの他のハンドラはマルチプレクシング・ディスパッチ機能を示し、どのように実装するかに応じて、これらは同じ機能にすることが可能である。言い換えれば、以下で詳しく説明するように、各タイプの入出力ハンドラ参照（例えば、読取り、書込み、デバイス制御など）はデバイス・オブジェクト内のエクステンション・データとファイル・オブジェクト内のコンテキスト情報を使用して、メッセージを適切なハンドラへルーチングするマルチプレクシング・ディスパッチ機能を指している。バリデーション・テーブルを参照するデバイス・オブジェクト内のエクステンション・データは、作成オペレーションで親ファイル・オブジェクトの指定がないとき使用される。指定があれば、親ファイル・オブジェクトのコンテキスト情報は正しいバリデーション・テーブルを示している。

【００７８】図５は、ドライバ開発者によって所望により利用でき、ドライバ専用情報を含んでいる特定のデバイス・エクステンション・エリア８２をもつドライバ・オブジェクト８０を示している。ドライバ・オブジェクト８０のデバイス・エクステンション・エリア８２内の定義されたロケーションには、ファイル・タイプ・バリデーション・テーブル８４と呼ばれ、ファイル・オブジェクト・タイプ８６のストリング表現を含んでいるデータ構造への参照と、表現される各ファイル・タイプ別の関連する作成ハンドラ８８への参照が置かれている。作成マルチプレクシング・ディスパッチ機能はファイル・タイプ・バリデーション・テーブル８４を使用して、作成されるファイル・オブジェクト・タイプを検査し、そのあと、コントロールを適切な作成ハンドラへ引き渡す。これについては、以下の図８の説明個所で詳しく説明する。検査されるストリングはＩＲＰ作成要求に見出され、ユーザ・モードのNtCreateFile関数コールと共に使用されるファイル名ストリングから創られる。NtCreateFile関数コールは、ピン・インスタンスまたは他のメカニズムを作成するためにユーザ・モード関数セルの中で行われる。

【００７９】図６は、ソフトウェア・ドライバ開発者が使用するために解放されているファイル・コンテキスト・エリア９２をもつファイル・オブジェクト９０を示している。参照はファイル・コンテキスト・エリア９２からＩＲＰ要求ハンドラ・テーブル９４へ行われる。ＩＲＰ要求９６の異なるタイプは特定のハンドラ９８と関連づけられ、適切なマルチプレクシング・ディスパッチ機能はこの情報を使用して正しいハンドラにアクセスする。正しい作成ハンドラを決定する場合には、ファイル・タイプ・バリデーション・テーブル１００と呼ばれるデータ構造が参照され、そこには、ファイル・オブジェクト・タイプ１０２のストリング表現と、表現される各ファイル・タイプ別の関連する作成ハンドラへの参照１０４が入っている。子ファイル・オブジェクト（つまり、デバイス・オブジェクトではなく別のファイル・オブジェクトを親としてもつファイル・オブジェクト）の場合は、タイプはファイル・オブジェクト・タイプ１０２内のストリングと比較されるストリングで表されている。一致するものが見つかると、関連する作成ハンドラは、参照１０４のうち、一致したファイル・オブジェクト・タイプ・ストリングと関連づけられている参照を使用してアクセスされる。一致するものが見つからなければ、要求は無効であるので、エラー表示が出されることになる。

【００８０】次に、図７を参照して説明すると、図は作成バリデーションとメカニズムをセットアップするためのインストレーション・プロシージャを示している。ステップ１０６で、インストール・プログラムは適切なマルチプレクシング・ディスパッチ機能への参照をドライバ・オブジェクトの中に作成する。図４に示すように、作成ハンドラは汎用マルチプレクシング・ディスパッチ機能を指している。同様に、ドライバ・オブジェクト７６の中の他のすべてのハンドラ参照は、特定ハンドラと密接な関係をもつ他の汎用(generic) マルチプレクシング・ディスパッチ機能を必要に応じて指している。別の方法として、各ハンドラ参照は同一マルチプレクシング・ディスパッチ機能を指すことも可能であり、その場合は、このディスパッチ機能はＩＲＰ要求を処理したあ

と、それを適切なハンドラへルーチングすることができる。この方法によるマルチプレクシング機能は異なる種類の要求（例えば、作成、書込みなど）を受け付ける必要があるため、複雑化することが避けられない。

【００８１】次に、ステップ１０８で、ソフトウェア・ドライバ実行可能コードのインストールの一部として作成された各デバイス・オブジェクトは、図５に示すようにファイル・タイプ・バリデーション・テーブル８４を参照するように調整される。最後に、ステップ１１０で、ＩＲＰ要求の処理は、適切なデバイス・オブジェクト８０から参照されたファイル・タイプ・バリデーション・テーブル８４を使用してマルチプレクシング・ディスパッチ機能から開始される。

【００８２】ファイル・オブジェクトが作成されると、適切なＩＲＰディスパッチ・テーブル９４が作成され、必要ならば、インデックスされたファイル・オブジェクト・タイプ・バリデーション・テーブル１００と一緒に参照される。ファイル・オブジェクト・タイプ・バリデーション・テーブルの作成は、ファイル・オブジェクト・タイプに従って用意された作成ハンドラ内で行われる。データ構造が作成され、これはＩＲＰディスパッチ・テーブル９４とファイル・オブジェクト・タイプ・バリデーション・テーブル１００を表しており、それを指す参照は作成される特定ファイル・オブジェクト９０のファイル・コンテキスト情報９２と一緒に特定ロケーションにストアされる。

【００８３】次に、図８を参照して説明すると、図は作成マルチプレクシング・ディスパッチ機能のオペレーションとそのバリデーション・メカニズムを示すフローチャートであり、そこには、システム・ドライバ・オブジェクト、デバイス・オブジェクト、およびファイル・オブジェクトから参照されるデータ構造との相互作用も示されている。ステップ１１２で、ユーザ・モード・プロセスはファイル・オブジェクトを作成する入出力要求を送信する。この入出力作成要求はNtCreateFileのシステムＡＰＩを呼び出すことによって行われる。ステップ１１４で、入出力マネージャはドライバ・オブジェクト７６内の参照に基づいてＩＲＰをマルチプレクシング・ディスパッチ機能７８に送信する（図４参照）。

【００８４】マルチプレクシング・ディスパッチ機能７８が作成要求のＩＲＰを受け取ると、ステップ１１６でテストが行われ、親ファイル・オブジェクトがあるかどうかが判断される。この判断を行うために必要な情報はＩＲＰ自体の中にあるが、これはユーザ・モード・プロセスによって元来用意されたものである。ユーザ・モード・プロセスは「親」ファイル・オブジェクトを参照するハンドルを作成要求の一部として用意し、ＮＴシステムは「親」ファイル・オブジェクトへの正しい参照をもつＩＲＰを作成する。

【００８５】親ファイル・オブジェクトがなければ、右へのブランチがとられ、マルチプレクシング・ディスパッチ機能７８は適切なデバイス・オブジェクト８０からのデバイス・エクステンション８２を使用して、ファイル・タイプ・バリデーション・テーブル８４をステップ１１８で参照する（図５参照）。バリデーション・テーブル８４を使用して、マルチプレクシング・ディスパッチ機能７８は要求の中のストリングをファイル・オブジェクト・タイプ８６のストリングと比較することによって、ステップ１２０でファイル・オブジェクト・タイプを検査する。

【００８６】ステップ１２２でストリングが一致していると判断されると、適切な作成ハンドラがステップ１２４でアクセスされる。一致していなければ、作成要求はステップ１２６で拒否される。ステップ１２４でアクセスされた作成ハンドラはステップ１２６でファイル・オブジェクトを作成するか、あるいは作成させる。作成されたファイル・オブジェクトを使用して、適切な作成ハンドラは、以前に作成していたＩＲＰディスパッチ・テーブル９４を指すファイル・オブジェクト参照をファイル・コンテキスト９２の中に作成する。

【００８７】再びステップ１１６に戻って、親ファイル・オブジェクトが存在するかどうかが判断される。親ファイル・オブジェクトが存在するとステップ１１６で判断され、それが作成要求に関連するＩＲＰに入っていれば、マルチプレクシング・ディスパッチ機能７８は親ファイル・オブジェクト９０からのファイル・コンテキスト９２を使用して、ステップ１３０でＩＲＰディスパッチ・テーブル９２を参照する（図６）。作成要求の場合は、マルチプレクシング・ディスパッチ機能７８はステップ１３２でファイル・タイプ・バリデーション・テーブル１００を参照する。ファイル・タイプ・バリデーション・テーブル１００を使用して、マルチプレクシング・ディスパッチ機能７８は、上記と同じように、要求の中のストリングをファイル・オブジェクト・タイプ１０２のストリングと比較することによってステップ１３３でファイル・オブジェクト・タイプを検査する。

【００８８】ストリングが一致しているとステップ１３４で判断されると、適切な作成ハンドラがステップ１３８でアクセスされる。一致していなければ、作成要求はステップ１３６で拒否される。適切な作成ハンドラを使用して、ファイル・オブジェクトは１４０で作成され、作成ハンドラは新しく作成されたファイル・オブジェクトの新しいＩＲＰディスパッチ・テーブル９４を作成し、新しく作成されたＩＲＰディスパッチ・テーブル９４を指す参照を新しく作成されたファイル・オブジェクト９０のファイル・コンテキスト・エリア９２の中にステップ１４２で作成する。注意すべきことは、親ファイル・オブジェクトおよび有効に作成された子ファイル・オブジェクトとの相互作用を説明するために、どちらの場合も、図６に示す同じファイル・オブジェクト構造が

使用されていることである。どちらの場合も同じ構造が存在するが（新しいファイル・オブジェクトが作成されたあと）、これらはその使い方が異なり、含んでいる情報も異なっている。

【００８９】接続ピン・インスタンスが作成されるといつでも、接続ピンＩＤが引き渡され、これはピン・インスタンスの作成を「サポート」するファイル内のピン・ファクトリを示している。この分野の精通者ならば理解されるように、接続ピンＩＤは、ファイル・オブジェクトが検査されるのとまったく同じように、バリデーションテーブルでストリングとして検査することも可能であり、また、その実装方法もさまざまである。

【００９０】異なるドライバ間の接続を行うためには、あるドライバがそのような相互接続をサポートしていることを確かめるための共通メカニズムが必要である。この共通メカニズムは、接続ピン・ファクトリ機能を含むフィルタ機能を明らかにすることを可能にするものでなければならない。さらに、この種のメカニズムは、ドライバ開発者の柔軟性を向上するように拡張可能であることも必要である。

【００９１】コンプライアント・ドライバを定義し、機能を明らかにすることを可能にするために本実施の形態で選択されている１つのメカニズムは関連アイテムの「集合」と名づけられている。これは既存の入出力通信メカニズムと一緒に使用すると便利なメカニズムである。集合は集合全体を特定するＧＵＩＤ（グローバルにユニークな識別子）と集合内の各機能エレメントのＲＵＩＤ（相対的にユニークな識別子、例えば、集合自体内の相対的な識別子）を持つものとして論理的に定義されている。集合識別子および選択されたＲＵＩＤアイテムと一緒にオペレーションするために必要な他のデータ構造は、フィルタ・ハンドルをパラメータとして使用して入出力制御コールの一部として引き渡される。機能の完全なシステムを実装するためには、少数のＩＯＣＴＬを割当てるだけで十分である。実装されるとき、３つの異なる種類の集合がその機能に応じて確立されるので、必要になるＩＯＣＴＬは総計４個である。他の実装では、集合の使い方が異なる場合がある。特定のＩＯＣＴＬは、選択されたエレメント（ＲＵＩＤを使用する）をなんらかの方法で解釈または使用するように入出力制御のハンドラに通知する。さらに、制御フラグをＧＵＩＤおよびＲＵＩＤと一緒に渡して、制御情報をもっと詳しく指定することができる。

【００９２】最初の集合タイプはプロパティ集合であり、これはドライバ内または関連ハードウェア上に置かれている値または設定値と一緒に使用される。そのような設定値の例としては、転送レート、ボリューム・レベル（音量）などがある。１つのＩＯＣＴＬがプロパティ集合と関連づけられ、制御フラグは"get" プロパティ・コマンドと"set" プロパティ・コマンドとを区別している。このようにすると、同じデータ構造が特定のプロパティを設定または取得するように使用でき、ドライバは必要なアクションを使用されたＩＯＣＴＬに基づいて決定することができる。正しいプロパティは、ユニークなＧＵＩＤとＲＵＩＤの組合せからなる集合識別子(set identifier)によって特定される。

【００９３】メソッド集合は使用されるもう１つの集合タイプであり、これはドライバによって実行できるアクションの集合である。メソッド集合を特定するために必要なＩＯＣＴＬは１つだけであり、アクチュエートされる正しいメソッドは集合識別子のユニークなＧＵＩＤとＲＵＩＤの組合せによって特定される。メソッドはドライバを制御するために使用され、ドライバを使用するための初期化、バッファのクリアといった機能を含んでいる。

【００９４】イベント集合は、デバイス変更通知、データ・スターベーション通知などのドライバ処理や、ユーザ・モード・アプリケーションで使用すると便利である集合によって定義された他の通知に関連するイベントを管理するために使用される。２つのＩＯＣＴＬが使用され、１つは特定のイベントをイネーブルにするためのものであり、もう１つは特定のイベントをディスエーブルにするためのものである。ＲＵＩＤで識別された、与えられたイベントに必要なデータ構造は、イベントをイネーブルにするか、ディスエーブルにするかに関係なく共用することができる。

【００９５】集合を使用するには、入出力制御オペレーションは特定のＩＯＣＴＬおよび集合ＧＩＤ、ＲＵＩＤをもつデータ構造と他の必要なデータ（例えば、制御フラグ、データ構造など）を指す参照を使用して開始される。例えば、サウンド・カード・ドライバでボリューム・プロパティを設定することは、入出力制御オペレーションをプロパティ集合ＩＯＣＴＬ、集合プロパティ・オペレーションを示す制御フラグ、ボリューム設定値をもつプロパティ集合の適切なＧＵＩＤ、ボリューム・プロパティを示しているその集合内の特定ＲＵＩＤ、および新しいボリューム設定値を使用することを必然的に伴なう。

【００９６】サポートされる集合についてタイプ別に照会するには、ヌルＧＵＩＤとサポートされる集合の列挙を示す制御フラグとをもつ、特定の集合タイプのＩＯＣＴＬ（例えば、プロパティＩＯＣＴＬ、メソッドＩＯＣＴＬ、またはイベント・イネーブルＩＯＣＴＬ）が入出力コマンドの一部として出され、サポートされる集合ＧＵＩＤのリストが返される。ある集合内のサポートされるプロパティ、メソッドまたはイベントについて照会するには、集合ＧＵＩＤ、集合タイプＩＯＣＴＬ、ヌルＲＵＩＤ、およびサポートされるエレメントの列挙を示す制御フラグが入出力オペレーションと共に使用される。サポートされるＲＵＩＤのリストは入出力オペレーショ

ンの結果として返される。このリストから、サード・パーティ・エージェントは、実装されている集合のどのオプション・エレメント（もしある場合）がサポートされるかを判断することができる。

【００９７】ＧＵＩＤでユニークに識別された集合の仕様書は、ドライバ開発者とサード・パーティ制御エージェントのどちらもが実装ガイドとして利用できるメカニズムをドキュメント化している。サード・パーティ開発者は、あるドライバの機能を照会に対する応答に基づいて知り、事前プログラムされた知識を抽象的集合定義に基づいて知ることになる。同様に、ドライバ開発者は、知った機能をどのようなサード・パーティ・エージェントに対して提供する集合または集合グループを実装するときのガイドとして抽象的集合定義を使用することができる。

【００９８】ここで説明している接続機能を提供するためには、コンプライアント・ドライバはある種の集合をサポートしていなければならない。以下の表は、プロパティ集合フォーマットでサポートされ、本発明を実装するときに使用できるいくつかの重要な情報を示している。最初の表は、フィルタによって実装される接続ピン・ファクトリに関するプロパティに関し、２番目の表は、特定の接続ピン・ファクトリをテンプレートとして使用して作成される実際の接続ピン・インスタンスに関するプロパティを示している。

【００９９】

【表１】

（表１）

フィルタ・プロパティとその使い方	
プロパティ	説明
接続ピン・ファクトリ	特定のフィルタで作成できる異種タイプの接続ピン・インスタンスをリストしている。各区別可能なタイプはピン・ファクトリと呼ばれる。なお、これはこのデバイスでインスタンス生成できる接続ピン・インスタンスの総数ではなく、オーディオ入力やオーディオ出力のように、ユニークな接続ピン・タイプの数である。
接続インスタンス	ある接続ピン・ファクトリの、すでに作成されたインスタンスの数と、その特定接続ピン・ファクトリ用にサポートされるインスタンスの最大数をリストしている。フィルタが実際に接続されるまで総数が決定できないときは、このプロパティはａ－１を返す。
データ・フロー	接続ピン・ファクトリがフィルタに対して取り得るデータ・フローの方向をリストしている（例えば、フィルタに入る方向、フィルタから出る方向、またはフィルタから出入りする方向）。

【０１００】

【表２】

（表１のつづき）

通信	ある接続ピン・ファクトリの通信要求をＩＲＰの処理の観点からリストしている。一部の接続ピン・ファクトリは相互接続できないが、グラフ上のソース・ポイントを表すデータのファイル・ソースへの「ブリッジ(bridge)」のように、関連づけられた他の形態の制御メカニズムを有している。ブリッジ制御メカニズムは情報がストアされているファイル名を間接的に設定することを可能にする。 実施の形態では、エージェント（接続ピン・インスタンスを作るときどのピン・ファクトリを使用するかを判断する）は接続ピン・ファクトリの「なし」、「シンク」または入力、「ソース」または出力、「両方」および「ブリッジ」通信タイプの本来の意味を理解できなければならない。例えば、ソース接続ピン・インスタンスはシンク接続ピン・インスタンスなどへのハンドルまたは参照を必要とする。 通信タイプのコンテキストでは、シンクとソースとはＩＲＰを処理するときの接続ピン・インスタンスの処置に関する。シンクは処理するＩＲＰを受信するのに対し、ソースはＩＲＰを次の適切な処理コンポーネント上に引き渡す。 シンクでもソースでもなく、接続グラフのエンド・ポイントを表している通信タイプが２つある。エンド・ポイントはデータが接続されたフィルタに出入りする場所を表している。「なし」とは接続タイプがインスタンシェイトできないことを意味するのに対し、ブリッジ通信タイプとは特定のプロパティを操作できるようにインスタンシェイトできるエンドポイントを意味する。例えば、ファイル・リーダの一部であるブリッジ・エンドポイントは処理されるデータをストアしているファイルのパスとファイル名を含んでいるプロパティをもっている可能性がある。

【０１０１】　　【表３】

（表１のつづき）

データ範囲	接続ピン・ファクトリがサポートできる、可能な限りのデータ範囲をリストしている。関連すれば、データのフォーマットも含まれる。一実施の形態では、データ範囲の配列があとに続くカウントは接続ピン・タイプがサポートできるが、プロパティの一部として使用される。この実装においては、異なるデータ範囲が異なるメディアまたはインタフェースの下でサポートされる場合（下記参照）、異なる接続ピン・ファクトリが特定フィルタで利用でき、この相違を受け入れることができる。さらに、各データ範囲構造はビット数やチャネル数といった、フォーマット固有の詳細用に拡張可能である。接続ピン・インスタンスが使用する実際のデータ・フォーマットはインスタンスの作成時に設定される。データ範囲プロパティは、その実際のデータ・フォーマットが特定の接続ピン・インスタンス用にどのようにされるべきかを判断するとき使用され、サード・パーティ制御エージェントによってアクセスまたは照会される。
インタフェース	特定の接続ピン・ファクトリ上のサポートされるインタフェースを示す他の集合ＧＵＩＤをリストしている。インタフェースは接続ピン・ファクトリを通して通信できるタイプまたはデータのタイプである。例えば、ＭＩＤＩデータ、ＣＤミュージック、ＭＰＥＧビデオなどは、フィルタが処理できる特定の規則とフォーマットをデータがもっているという意味でインタフェースとなる。このようなインタフェースはデータを発信するためのプロトコルも含んでいる。インタフェースはそれが通信されるときの媒体から独立している。
媒体	特定の接続ピン・ファクトリ上のサポートされる媒体をリストしている。媒体とは、ＩＲＰベース、ソケットなどのように、情報が転送されるときの方法またはメカニズムである。インタフェースは種々の異なる媒体の上に定義できる。本明細書で説明している好適実施の形態と実装例では、ＩＲＰベースの媒体とファイル入出力ベースの媒体が使用されている。

【０１０２】

【表４】

（表１のつづき）

データ交差	データ範囲のリストが与えられているとき接続ピン・ファクトリによって作られた最初の受付可能または「最良」データ・フォーマットを返す。このアプローチは、サード・パーティ・エージェントが異なるフィルタを１つにチェインニングするときデータ要件を判断できるようにするために使用される。--実装例では、データ交差(data intersection) プロパティはデータ範囲のリストの制約が与えられているとき接続ピン・ファクトリによって作られた最良データ・フォーマットを判断するために使用される。データ範囲のリストは前述したように接続される別のピン・ファクトリでデータ範囲プロパティを使用して取得できる。 サード・パーティ制御エージェントは、データ・タイプの詳細を知らないので、ある接続ピン・ファクトリのデータ範囲リストを使用し、現行接続ピン・ファクトリで「最良」（例えば、最初の受付可能データ・フォーマット）を返すことができる。２つの交差接続ピン・ファクトリの範囲のセットを返すことができるが、最良フォーマットだけがドライバによって返される。このようにすると、サード・パーティ制御エージェントはこの「最良」データ・フォーマットをグラフ内の次のドライバに適用して、仮想的な(virtual) 接続集合を作成してから、実際に接続ピン・インスタンスの作成を開始し、フィルタのグラフ全体を１つに接続することができる。これにより、制御エージェントはユーザによって選択された特定フィルタ・グラフの実行可能性を評価し、ユーザに起こる可能性のある問題を指摘してから、実際にグラフを接続することができる。返されるデータ・フォーマットはフィルタですでに行われている接続が与えられているとき、使用可能なフォーマットで制限することもできる。 このプロパティは実際の接続が行われるまで特定のデータ・フォーマットが決定できないときや、交差が異なる接続ポイント上の複数のデータ・フォーマットに依存するときエラーを返すことができる。基本的には、交差情報が提供され、プロパティ自身はデータ・フォーマットを返す。

【０１０３】　　【表５】

（表２）

接続ピン・インスタンス・プロパティとその使い方	
プロパティ	説明
状態	接続ピン・インスタンスの現状態を記述している。起こり得る状態には、停止、データ取得、データ処理、休止またはアイドルなどがある。状態は接続ピン・インスタンスの現モードを表し、ドライバの現在の機能とリソース使用状況を判断する。停止状態は接続ピン・インスタンスの初期状態であり、最小リソース使用状況のモードを表している。停止状態は、実行状態に到達するためにデータ処理のレイテンシ(latency) が最大であるポイントも表している。取得状態は、データがこの状態で転送されない場合でもリソースが取得されるモード（バッファ割当てなど）を表している。休止状態は最大リソース使用状況のモードと、実行状態に到達するまでの処理レイテンシがこれに応じて低いことを表している。データはこの状態で、転送または「プリロール(prerolled) 」できるが、これは実際には実行状態ではない。実行状態はデータが接続ピン・インスタンスで実際に消費または作成される（例えば、転送され、処理される）モードを表している。フィルタと基礎となるハードウェアの目的に応じてカスタム・プロパティを使用すると、コントロールの解像度を向上することができる。例えば、外部レーザ・ディスク・プレイヤをスピンアップさせるには、そのクラスに固有のある種のカスタム「モード」プロパティを設定することになる。このプロパティをセットすると、モードの効果に応じてデバイスの状態も変更されるが、必ずしもそうとは限らない。

【０１０４】　　【表６】

（表２のつづき）

優先度	フィルタによって管理されるリソースへのアクセスを受け取るための接続ピン・インスタンスの優先度を記述しており、リソース割当ての調停(arbitration)においてフィルタによって使用される。このプロパティがサポートされていれば、サード・パーティ制御エージェントは限定リソースをこの特定接続およびインスタンスと共用できる他のすべてのドライバの他のすべての接続ピン・インスタンスに相対的な特定ピン・インスタンスの優先位置を指定することができる。この優先度プロパティが実装されていると、エージェントは単一優先度クラス内の優先度をもっときめ細かくチューニングすることができる。例えば、優先度はある種のサブクラスを関連づけることができる。同一リソースを競合する２つのドライバが同一優先度クラスをもっていれば、サブクラス優先度は２ドライバ間のリソース割当てを決定するために使用される。サブクラス優先度も同一であれば、調停により、最初の接続ピン・インスタンスがリソースを受け取ることになる。
データ・フォーマット	接続ピン・インスタンスのデータ・フォーマットを取得または設定するために使用される。

【０１０５】上記の表は単なる例示であり、これに限定されるものではない、この分野の精通者ならば理解されるように、異なるドライバ間の接続を作成するためには多数の異なるプロパティとスキーマを実装することが可能である。１つの重要な要素は標準化係数(standardization factor)であり、異なるドライバ製造者または開発グループは同一のプロパティ集合を実装する能力をもっているので、相互接続できるドライバを作成できるようにすることである。

【０１０６】もう１つの有用なプロパティ集合は、あるフィルタ上の入力と出力の接続ピン・ファクトリの内部関係に関するトポロジ情報を与える。この情報はあるフィルタ上の入力ピン・ファクトリおよび対応する出力ピン・ファクトリの関係だけでなく、入力と出力のピン・ファクトリの間でどのようなタイプの処理が行われるかも記載している。行われる処理の例としては、異なるデータ変換、データ伸張、エコー打消し(cancellation)などがある。このような情報は複数のフィルタを使用して仮定上の接続パスをトレースしてから実際の接続ピン・インスタンスおよび接続を作成する自動化フィルタ・グラフ作成機能で使用すると、便利である。基本的には、トポロジ情報はフィルタの内部構造を説明し、これをプロパティ集合メカニズムを通してサード・パーティ・エージェントからの照会に公開する。

【０１０７】従って、コンプライアント・ドライバは指定されたプロパティ集合を実装しているものにすぎない。そのため、サード・パーティ制御エージェントは、あるプロパティ集合がサポートされていることが分かったとき、コンプライアント・フィルタに対し照会と設定を行うことができる。全体目標は、異なるフィルタを１つに接続してフィルタ・グラフを作る方法に関する十分な情報を得ることである。

【０１０８】汎用集合メカニズムを使用すると、最小限の機能を実装してコンプライアント・ドライバのシステムをサポートできるが、その場合でも、拡張性は無制限である。集合は、特定の集合が実装されている限り、相互操作可能で相互接続可能なドライバのシステムを作成するために多数の異なるドライバ開発者によって独立にコーディングできる仕様書の中で定義することができる。さらに、この仕様書には、サポートしなければならない必須のプロパティ、メソッド、およびイベントを定義できるだけでなく、ドライバ機能と拡張機能に応じて実装できるオプションのプロパティ、メソッド、およびイベントも定義できる。最小限に要求される基本的共通性のほかに、ドライバ開発者は独自の集合を定義し、これらにＧＵＩＤを割当てることにより追加の機能を組み入れることも可能である。

【０１０９】次に、図９および図１０を参照して説明する。図は２つのカーネル・モード・フィルタを接続するプロセスを図形化して示している。図９はロジック・ブロック説明図であり、そこでは各フィルタ・インスタンスと接続ピン・インスタンスはファイル・オブジェクトで表されている。図１０はファイル・オブジェクトおよび適切な接続を作成するときのステップを示すフローチャートである。

【０１１０】ステップ１４４からスタートして、フィル

タＡ１４６のインスタンスおよびフィルタＢ１４８のインスタンスはユーザ・モード・エージェントによって作成される。これらは特定のデバイスでファイルを作成する標準ファイル・システムＡＰＩを使用して作成される。フィルタＡ１４６とフィルタＢ１４８がコンプライアント・フィルタまたはドライバとなっているのは、これらが適切なプロパティ、メソッド、およびイベント集合を実装して接続ピン・インスタンスの作成をサポートし、サポートされる集合とそのフィルタ用に定義された接続ピン・ファクトリに関してそれぞれのフィルタの機能に照会するようになっているからである。

【０１１１】サード・パーティ制御エージェントは、ステップ１５０でフィルタＡ１４６とフィルタＢ１４８にそれぞれ照会し、利用可能な接続ピン・ファクトリおよびそれから作成できる接続ピン・インスタンスの属性を判断する。これらの属性には、前述したように、各フィルタ１４６と１４８のそれぞれのピン・ファクトリのタイプ別の接続フォーマットとデータ・フォーマットがある。照会は、以下で詳しく説明する集合ベース照会メカニズムを使用して行われる。

【０１１２】上記情報の照会を終えると、サード・パーティ制御エージェントは、以前に照会したデータ・フォーマットと接続フォーマットの範囲に基づいて最適接続フォーマットを決定する。この決定はステップ１５２で行われ、選択した接続パスの要求事項に応じて同じフィルタをサード・パーティ・エージェントが異なった使い方をできるようにする。サード・パーティ制御エージェントはデータ交差プロパティ(data intersection property)、トポロジ情報、および接続ピン・ファクトリを両方のフィルタで使用して、作成される実際のフィルタ・グラフに応じてデータ・フォーマットと接続配置の最良の選択方法を決定する。

【０１１３】入力フィルタ・ピン・インスタンス１５４はステップ１５２で判断された最適検出情報を使用して、ステップ１５６でサード・パーティ・エージェントによって作成される。入力ピン・インスタンス１５４はファイル・オブジェクトであるので、ハンドルが作成プロセスから返されるが、これは入出力要求を入力インスタンス１５４へ送るために使用できる。さらに、入力ピン・インスタンス１５４の作成が有効性の検査をされているが、この作成には、図４～図６、図７および図８を参照して前述したルーチングおよびバリデーション・メカニズムが使用される。

【０１１４】接続を完成するために、出力ピン・インスタンス１５８は、以前に作成された入力ピン・インスタンス１５４のハンドルをNtCreateFileコールの中でパラメータとして使用してステップ１６０で作成される。出力ピン・インスタンス１５８がこのようにして作成される効果として、システム・ファイル管理機能と入出力管理機能を使用して内部ＩＲＰスタック構造が作成される。このスタック構造により、オリジナルの書込みコマンドは、さまざまな方法で接続された接続ピン・インスタンスとフィルタによって適当な順序で連続処理することが可能になるので、異なるフィルタ間の直接データ・フローが容易化される。これを行うためには、入力ピン・インスタンスを供給する関連出力ピン・インスタンスの前に入力ピン・インスタンスが作成されていることが必要である。

【０１１５】デバイス・オブジェクトのスタック深さパラメータは、このドライバに送られるＩＲＰ用にスタック・ロケーションをいくつ作成するかを制御する。スタック深さパラメータは、デバイス・オブジェクトが初めて作られるときは１つであると想定されているが、複数のドライバが１つにチェイニングされているかどうかに応じて、あとで変更することも可能である。現システムでは、この変更は、必要ならば、出力ピン・インスタンスが初期「停止」状態から「取得」または他の状態に遷移するとき行われる。接続ピン・インスタンスの状態遷移はＩＲＰを正しく作成し、処理するための正しいスタック深さパラメータ情報を決定するメカニズムである。

【０１１６】チェイニングされた接続ピン・インスタンス集合を正しく割当てるためには、接続ピン・インスタンスを特定の順序で停止状態から出るように遷移させる必要がある。つまり、最後の入力ピン・インスタンス（この例では、入力ピン・インスタンス１５４）から始まって、関連の（例えば、接続された）出力ピン・インスタンス（この例では、出力ピン・インスタンス１５８）まで連続的に逆方向に戻っていく必要がある。多数のフィルタが１つにチェイニングされていれば、最も深いフィルタまたはブリッジの入力ピン・インスタンスが遷移の開始ポイントであって、ブリッジまたはフィルタ上の初期出力ピン・インスタンスが設定されるまで逆方向に連続的に構築していかなければならない。言い換えれば、停止状態から出る遷移はチェインを逆昇るように行われ、各接続ピン・インスタンスが前の接続ピン・インスタンスの後で必要になるスタック・サイズを得るようにしなければならない。代表例として、必ずしもそうする必要はないが、接続ピン・インスタンスは停止状態から取得状態へ遷移するが、以下での説明の便宜上、取得状態への遷移はスタック深さパラメータの調整に関しては、停止状態から出る遷移と同じ目的を果たすようになっている。

【０１１７】すべてのピン・インスタンスが取得状態になったあとは、ストリーム読取りと書込みをフィルタ・グラフに対して出すことができる。なお、注目すべきことは、ここで説明しているシステムが、関連入力と出力ピン・インスタンスの接続をどの順序でも行うことができ、停止状態からの遷移だけはボトムアップ方式、つまり、最も深いものから先に行わなければならないことである。さらに、フィルタ・グラフは初期作成後に変更が

行えるように再構成可能になっている。変更が行われるときは、状態遷移は停止状態にある接続ピン・インスタンスだけで行い、正しいスタック深さパラメータ情報が得られるようにする必要がある。

【０１１８】フィルタ上に見出される接続ピン・ファクトリは、フィルタが特定フォーマットでデータを消費および／または作成できる場所を表している。例えば、特定の接続ピン・ファクトリは、１６ビットの４４キロヘルツＰＣＭオーディオや８ビットの２２キロヘルツＰＣＭオーディオなどの、いくつかの異なるデータ・フォーマットをサポートすることができる。前述したように、接続ピン・ファクトリやそのデータ・フォーマットなどの異なる機能については、適切なプロパティ集合メカニズムとシステム入出力機能を使用してフィルタから照会することができる。実際の接続ピン・インスタンスはピン・ファクトリから受信した情報に基づいて作成される。

【０１１９】単一のストリーム書込みまたはストリーム読取りオペレーションがユーザ・モード・エージェントから出されると、接続されたフィルタを通してデータの連続処理が行われるストリーミング環境では、ＩＲＰ制御用の２つのメイン・メソッドがＮＴオペレーティング・システムのネーティブ機能の一部として使用することができる。第１のメソッドでは、個別ＩＲＰは各フィルタによって作成され、次のフィルタへ送られて処理され、このフィルタは新しいＩＲＰを作成し、チェインを下って次々と処理されていく。他のメソッドでは、単一のＩＲＰが使用され、入出力マネージャと相互作用するために用意された標準プロシージャを使用して連続フィルタ間で受け渡しされていく。チェイン上の各連続フィルタごとに新しいＩＲＰを作成していく第１のメソッドが使用される場合は、フィルタ間の相互接続順序が重要でないのは、フィルタはＩＲＰのデスティネーション(destination) だけを知っていればよく、入出力マネージャをコールしてＩＲＰを指定のフィルタへ送ることができるからである。ＩＲＰが再使用される場合に重要なことは、停止状態からの接続ピン・インスタンスの遷移が再使用ＩＲＰを受け取る最後のフィルタから始まるように行われ、再使用ＩＲＰを受け取る最初のフィルタまで、または処理のためにＩＲＰを作成したフィルタまで逆方向に処理していくことである。

【０１２０】相互接続カーネル・モード・フィルタの本実施の形態と実装例は、ＩＲＰ共用を利用してドライバ開発の複雑さを軽減し、より堅牢なドライバが作成されることを可能にし、処理を効率化するという利点がある。「ボトム・アップ」のピン・インスタンス状態遷移パスは、正しいスタック順序が連続ドライバによって処理されるＩＲＰに作成され、各ドライバ・オブジェクトが適切なスタック深さパラメータ集合をもつことを保証する。さらに、受信側入力ピン・ファクトリの現状態は、状態遷移シーケンスを正しくたどっていたかどうかを確かめるためにチェックされる。そのような理由から、特定の接続ピン・ファクトリの通信プロパティは起こり得るフロー方向を判断し、接続ピン・インスタンスの状態遷移を正しく分配する上で支援する。

【０１２１】出力ピン・インスタンス（またはＩＲＰソース）を作成するとき、別のフィルタ上の入力ピン・インスタンス（またはＩＲＰシンク(sink)）を表すファイル・オブジェクトへの参照はNtCreateFileコールの一部として引き渡される。適切な作成ハンドラはマルチプレクシング・ディスパッチ機能とデバイス・オブジェクト／ファイル・オブジェクト階層を使用して、前述したように実行される。この作成ハンドラは入力ピン・インスタンスをもつフィルタ（例えば、図９のフィルタＢ１４８）のデバイス・オブジェクトを、入力接続ピン・インスタンス・ファイル・オブジェクト（例えば、入力ピン・インスタンス１５４）を通してアクセスすることができる。デバイス・オブジェクトから、前のスタック深さパラメータを読み取ることができ、出力ピン・インスタンスをもつフィルタのデバイス・オブジェクトのスタック深さパラメータが増加される。例えば、フィルタＡ１４６に関連するデバイス・オブジェクトは、図９に示す接続ではフィルタＢ１４８に関連するデバイス・オブジェクトのそれから増加されるスタック深さパラメータをもっている。これは停止状態から出る遷移のとき行われるのが通常であり、ＩＲＰは接続ピン・インスタンスが停止状態にある間はルーチングされない。

【０１２２】フィルタがＩＲＰを処理するとき、その特定フィルタ用に指定された情報を含んでいる、ＩＲＰスタック内のどのスタック・フレームまたはロケーションをアクセスすべきかを、関連デバイス・オブジェクトのスタック深さパラメータを参照するか、あるいはそれを使用することによって知る。さらに、現フィルタは、デバイス・オブジェクトのスタック深さパラメータを減らして次のフィルタのＩＲＰスタック・ロケーションに位置づけることによって、処理チェイン上の次のフィルタ用のＩＲＰを準備する。

【０１２３】フィルタ・コードはＩＲＰスタック内の次のロケーションを準備し、入出力マネージャをコールしてＩＲＰを、指定された通りに次のフィルタに渡すことを受け持っている。このようにすると、フィルタは、特定の接続ピン・インスタンスを表す、どのファイル・オブジェクトがＩＲＰと関連データを受け取って処理するかを指定することができる。従って、ＩＲＰの順次処理のためにそれぞれのデバイス・オブジェクトをスタックするためのIoAttachDeviceのような標準入出力マネージャ・コールは使用されない。

【０１２４】注目すべきことは、接続ピン・インスタンス間の接続を作成することは、その接続を表すために新しいデバイス・オブジェクトを作成することを意味しな

いことである。基礎となる単一のデバイス・オブジェクトはフィルタのインスタンスとそのフィルタ上のすべての接続ピン・インスタンスをサポートするために使用される。正しいデータ処理のために必要な具体的情報はファイル・オブジェクトのコンテキスト・エリアに保存されていて、コンテキスト情報はそのまま残されているが、非ページ・メモリ使用量は最小限に保たれている。さらに注目すべきことは、ＩＲＰベースの媒体について説明してきたが、相互接続フィルタ間の通信のための他の媒体も使用できることである。そのようなものとして、非ホスト・ハードウェアとハードウェア間通信での直接関数コールがある。

【０１２５】次に、図１１と図１２および図１３を参照して、図１（従来技術）と図２（相互接続カーネル・モード・ドライバのハイレベル・ロジック図）に示すソフトウェア・ドライバの正しい作成、接続、および状態遷移順序について説明する。図１１はボックス１６２で囲んだロジック構造とそこに含まれる処理ステップを示している。図１２は接続ピン・インスタンスを作成して、カーネル・モード・フィルタの相互接続を完成する様子を示し、図１３に示すフローチャートにボックス１６４で囲んだ処理ステップを含んでいる。

【０１２６】すべての相互接続が行われている図１２の状態にあるとき、カーネル・モード・フィルタ・システムは処理を行うために読み書きの準備状態にある。入出力システムは正しい状態遷移プロセスによって正しく設定されたＩＲＰスタック情報を使用して、ストリーム読取りと書込みをそれぞれの接続ピン・インスタンスを通して異なるフィルタ・エレメントに引き渡す。なお、グラフを作成するために使用されたエージェント以外の、ある種の外部ソフトウェアは、ブリッジまたはフィルタ自体とハードウェアも含めて、ストリーム読取りとライトのデータを提供する。

【０１２７】ステップ１６８からスタートしたあと、制御エージェント１７０はステップ１８０で、リーダ・フィルタ１７２、デコンプレッサ・フィルタ１７４、効果フィルタ１７６、およびサウンド・レンダリング・フィルタ１７８のインスタンスを作成する。さらに、リーダ・フィルタ１７２とディスク・ドライバ１８２との間の接続が行われて、データがディスク・ドライブから持ち込まれるようにする。各フィルタ・インスタンスの作成は標準入出力コールを使用してデバイス入出力ディレクトリ階層に見出される適切なデバイス上のファイルをオープンすることにより、ユーザ・モード制御エージェント１７０によって行われる。このコールからは、各フィルタのインスタンスを表すファイル・オブジェクトへのハンドルが返される。

【０１２８】ステップ１８４で、サード・パーティ・エージェントは効果フィルタ１７２、デコンプレッサ・フィルタ１７４、効果フィルタ１７６、およびサウンド・レンダリング・フィルタ１７８に照会し、接続ピン・ファクトリの機能を判断する。これらの機能は、どのような種類の入出力ピン・インスタンスが作成できるか、特定のフィルタは各接続ピン・ファクトリのインスタンスをいくつサポートするか、各接続ピン・ファクトリでサポートされるデータ・フォーマット、通信バスの媒体またはタイプなどがある。これらの機能は以前に詳しく説明したプロパティ集合メカニズムを使用して照会され、カーネル・モード・フィルタは、適切な「集合」（例えば、プロパティ集合）をサポートしているのでアーキテクチャに従っていることが前提になっている。

【０１２９】ステップ１８４でのこのようなすべての照会情報は、チェインニングされた接続パスが、適切な接続ピン・インスタンスを作成し、接続することによりそれぞれのフィルタ間で可能であるかどうかを判断するために使用される。サード・パーティ・エージェントは、相互接続のために必要なピン・インスタンスのタイプを判断し、与えられた目的を達成するフィルタ・グラフを作成する。

【０１３０】サポートされるデータ・フォーマットに基づく接続フォーマットの決定はステップ１８６で行われる。フィルタ上のトポロジ情報、データ・フォーマット、およびデータ交差プロパティを使用すると、仮定上の(hypothetical)フィルタ・グラフを作成することができる。接続順序は重要でないので、そのようにする必要はないが、フィルタ・グラフを作成しようとするとき時間を節約することができる。この仮定上のフィルタ・グラフがエラーなしで作成されるのであれば、サード・パーティ・エージェントは、相互接続する接続ピン・インスタンスの作成を信頼して行うことができるという安心が得られる。実際のピン・インスタンスが作成されていなければ、ある種の照会からはエラーが返されるので、かかる接続ピン・インスタンスを作成してから仮定上のフィルタ・グラフを作成するようにすると、実行可能であるとの信頼できる指示が返されることになる。この場合も、仮定上のフィルタ・グラフは相互接続を行う前にテストすることが可能である。

【０１３１】正しい接続情報が分かっているとステップ１８６で決定されると、入力ピン・インスタンスを作成し、相互接続して、図１３のボックス１６４で囲まれた処理ステップのループで表すことができる。このループはデータ・ストリームのソースから最も遠くに離れた入力ピン・インスタンスから始まる処理ステップを含んでいる。この最後の入力ピン・インスタンスは「最も深い」ピン・インスタンスと呼ばれ、これは最初に作成され、そのあとに関連する出力ピン・インスタンスを続けることができる。従って、接続は、以前に作成された入力ピン・インスタンスのハンドルを使用して出力ピン・インスタンスを作成したものである。

【０１３２】このパターンは、すべての入力ピン・イン

スタンスが関連出力ピン・インスタンスとの接続の前に、それ以後連続して作成されるように続けられる。このような接続シナリオは単なる例示であり、それぞれの出力と入力ピン・インスタンスを接続して、本発明によるカーネル・モード・フィルタ間の接続を形成する他の可能な方法を限定するものではない。フィルタは、入力ピン・インスタンスからのハンドルが別のフィルタ上の接続された出力ピン・インスタンスの作成期間に使用される限り、実装に従ってどの順序でも接続することが可能である。さらに、前述したように、初期作成後（および使用後も）フィルタ・グラフに変更を行うことが可能である。

【０１３３】ループの最初の繰り返しでは、入力ピン・インスタンス１８８はステップ１９０で作成される。作成機能からハンドルを受け取ると、サード・パーティ制御エージェント１７０はそのハンドルをNtCreateFileコールでパラメータとして使用してステップ１９４で出力ピン・インスタンス１９２を作成する。最初の繰り返しでこれを行うと、サウンド・レンダリング・フィルタ１７８は、それぞれ対応する接続ピン・インスタンス１８８と１９２を通して効果フィルタ１７６に実効的に接続される。現実装では、NtCreateFileコールはユーザ・モード・クライアントが利用できるＡＰＩの関数コールの一部として「ラップ(wrapped)」される。これにより、サード・パーティ・エージェントのユーザ・モード開発者は詳細を知る必要から解放されるので、すべての関係機能を単一ユーザ・モードＡＰＩに集中させることができる。

【０１３４】ステップ１９６で、サード・パーティ・エージェントは作成すべき入力ピン・インスタンスが他にも残っているかどうかを判断する。残っていれば、入力ピン・インスタンスを作成し、そのあとに別のフィルタ上の対応する出力ピン・インスタンスを続けなければならない。最終的には、すべての接続が行われ、サード・パーティ制御エージェント１７０はストリーム化データ処理に対してフィルタ・グラフを準備する。

【０１３５】以上のようにして、入力ピン・インスタンス２０２はステップ１９０でボックス１６４で囲まれたループの２回目の繰り返しで作成され、他方、出力ピン・インスタンス２０４は入力ピン・インスタンス２０２のハンドルをステップ１９４でその作成の一部として使用する。最後に、この特定の例では、３回目と最後の繰り返しで、入力ピン・インスタンス２０６が作成され、そのあとに出力ピン・インスタンス２０８が続いて接続が完結する。

【０１３６】ステップ１９７で、サード・パーティ制御エージェント１７０は、フィルタ・グラフによるストリーム化データ処理に備えて各接続ピン・インスタンスを停止状態から取得状態へ遷移させる。それぞれのフィルタのデバイス・オブジェクトの各々でスタック深さパラメータを正しく設定するためには、状態遷移は「最も深い」または最後の接続ピン・インスタンス（例えば、処理のためのデータを受け取る最後の入力ピン・インスタンス）から始めて、最初の接続ピン・インスタンス（例えば、データをグラフに送り込む最初の出力ピン・インスタンス）に到達するまで相互接続カーネル・モード・フィルタのチェインを順次に「さかのぼる」ように行う必要がある。最初のフィルタまたはブリッジはＩＲＰを作成し、スタック・ロケーションはＩＲＰがグラフ内の各カーネル・モード・フィルタに効率よく連続して受け渡されるように十分に割当てられる。

【０１３７】最後に、サード・パーティ制御エージェント１７０はストリーム読取りと書込みを出し、ステップ１９８でデータを処理してからステップ２００で終了する。

【０１３８】前述したように、出力ピン・インスタンスの各作成には、そこに接続された入力ピン・インスタンスを表すファイル・オブジェクトのハンドルが必要である。このファイル・オブジェクト参照は、出力ピン・インスタンスの作成ハンドラが入力ピン・インスタンスに対応するデバイス・オブジェクトへの参照を、現在または将来のアクセスのためにセーブしておくことができるようにする。

【０１３９】もっと具体的に説明すると、これにより、入力ピン・インスタンスを管理するデバイス・オブジェクトのスタック深さパラメータは、停止状態から取得または他の状態へ状態遷移するとき出力ピン・インスタンスのドライバによってアクセスすることが可能になる。入力ピン・インスタンスに関連するスタック深さパラメータの値はアクセスされ、増加され、出力ピン・インスタンスに対応するデバイス・オブジェクトのスタック深さパラメータの中にセーブされる。

【０１４０】スタック深さパラメータは、共用ＩＲＰスタック構造のどこに特定フィルタ用のスタック・フレーム情報が置かれているかを判断するために使用されるが、これは各フィルタごとに異なっている。フィルタをこのように相互接続し、状態遷移を正しいシーケンスで行うと、通信をユーザ・モードにしなくても、カーネル・モードの相互接続フィルタのチェインを下るように単一ＩＲＰを受け渡してくことができる。

【０１４１】以上の説明から理解されるように、同一の接続ピン・ファクトリを基礎にした複数のインスタンスをもつことが可能である。例えば、オーディオ・ミキシング・フィルタは複数のピン・インスタンスをミックスして単一の出力ピン・インスタンスにしてから処理することができる。各入力インスタンスは同一タイプであり、フィルタは１つのタイプの入力ピンだけをサポートすることができる。このような構成の別の例として、複数の入力を１つの出力にする例がある。

【０１４２】逆のことを行うことも可能である。つま

り、スプリッタ・フィルタは単一入力接続ピン・インスタンスをもち、複数の出力ピン・インスタンスを提供することによりデータ・ストリームを倍にすることが可能である。この分野の精通者ならば理解されるように、上述してきた接続メカニズムから実際の実装とその要求条件に応じて種々態様に変形し、種々態様に組み合わせることが可能である。

【０１４３】ドライバ開発者が独立に実装できる共通メカニズム（例えば、プロパティ集合、メソッド集合、およびイベント集合）を、すべてのコンプライアント・フィルタにサポートさせることによって均一性と標準化が達成されているので、制御エージェントは異種ソフトウェア・プロバイダから提供されるコンプライアント・フィルタを都合よく接続することができる。さらに、接続ピン・ファクトリの観点から見た機能の多くはある状況では必要であっても、別の状況では必要でない場合がある。必要とされる接続ピン・インスタンスの決定は異なるフィルタ間の相互接続を実際に行うサード・パーティ制御エージェントによって最初に行われる。

【０１４４】次に、図１４を参照して説明すると、図は複数の処理コンポーネント間で使用されるときの、バッファ・アロケータ・メカニズムのオペレーションを示したものである。図には、特定の処理コンポーネントに関連するバッファ間のデータ・フロー（つまり、バッファ・フレーム間の実際のデータ転送）も示されている。コントロールは各処理コンポーネントに渡されるが、データは必要時にだけ転送され、一部の処理コンポーネントはデータ操作を実行し、データを既存バッファ・フレームに戻すようにしている。言い換えれば、データは新しいバッファに転送されることなく同じロケーションで処理されるので、「同所(in place)」で処理されると呼ばれている。

【０１４５】シンク処理コンポーネント２１０はバッファ・アロケータ・メカニズム２１２（四角で表されている）をその機能の一部としてもっている。バッファ・アロケータ・メカニズムは、サウンドまたはビデオ処理カード上のオンボード・メモリなどの特定のメモリにデータが置かれることを保証する必要のある処理コンポーネントや、前のバッファがバイト位置合わせ、フレーム・サイズなどの許容されない特性をもっているような処理コンポーネントにだけ存在している。さらに、バッファ・メモリのフレームを割当てるとき使用されるバッファ・アロケータ・メカニズムへの参照は円で示され、すべての処理コンポーネントはそのような参照をもっている。なお、ソース処理コンポーネント２１４は矢印２１８で示すようにシンク・バッファ・アロケータ２１２を参照するバッファ・アロケータ参照２１６をもっている。さらに、転送処理コンポーネント２２０はヌル（空白）のバッファ・アロケータ参照２２２をもち、シンク処理コンポーネント２１０も空の円２２３で示すようにヌルのバッファ・アロケータ参照をもっている。

【０１４６】この単純な処理例では、ソース処理コンポーネント２１４はシンクバッファ・アロケータ２１２を使用してバッファ・フレーム２２４ａを割当て、バッファ・アロケータ参照２１６を使用してアクセスされる。割当てられたフレーム２２４ａには、矢印２２６で示すようにソース処理コンポーネント２１４によってデータが満たされる。なお、ソース処理コンポーネントはある種のデータ操作または変換を行ってから、新たに割当てられたフレーム２２４ａにデータを書き込むことができる。

【０１４７】この時点で、シンク処理コンポーネント２１４は処理を終えており、処理のコントロールを矢印２２８で示すように変換処理コンポーネント２２０に引き渡す。バッファ・アロケータ参照２２２の参照がヌル値をもっており、バッファ・アロケータ・メカニズムが指示されていないので、変換処理コンポーネント２２０は、バッファの割当ても、データを一方のバッファから他方のバッファへ転送することも行わない。従って、変換処理コンポーネント２２０は矢印２３０で示すように、割当てられたバッファ・フレーム２２４ｂで「同所」のデータ変換を行う。

【０１４８】データは新しいバッファ・フレームに転送されていないので、バッファ・フレーム２２４ａ、フレーム２２４ｂ、およびフレーム２２４ｃは同じフレームであり、異なる処理コンポーネントへ連続して渡されるだけである。矢印２３１は割当てられたフレームがソース処理コンポーネント２１４と変換処理コンポーネント２２０の間で受け渡されることを示している。

【０１４９】最後に、変換処理コンポーネントは処理のコントロールを、矢印２３２で示すようにシンク処理コンポーネント２１０に引き渡す。なお、処理コントロールと一緒に、同じフレームがフレーム２２４ｂと２２４ｃの間で矢印２３４で示すように受け渡されて処理される。この場合も、図に示すように、フレーム２２４ａ、フレーム２２４ｂ、およびフレーム２２４ｃはいずれもソース処理コンポーネント２１４によって最初に割当てられた同じフレームであり、別々に示されているのは説明の便宜上である。

【０１５０】シンク処理コンポーネント２１０はデータの処理を終えると、矢印２３６で示すようにバッファ内の割当て済みフレーム２２４ｃを解放する。シンク処理コンポーネント２１０は最早バッファを使用していないので、矢印２３６は内側に向かってシンク処理コンポーネント２１０を指しており、このフレームは割当て解除することも再使用することも可能である。

【０１５１】図１５は、バッファ・アロケータ・メカニズムが上述してきた相互接続カーネル・モード・バッファ方式でどのように論理的に実装されるかを示している。図１４と図１５は共に同じフィルタ・グラフ・トポ

ロジを示し、バッファ・アロケータ・メカニズムのオペレーションを理解しやすくするために使用されている。関係ドライバとその部分はそれぞれ、ユーザ・モード・クライアントがドライバを制御することを可能にするアクセス・ポイントをもち、これらはファイル・オブジェクトで表されている。相互通信はＩＲＰを使用して行われる（ＩＲＰがカーネル・モード・ドライバによって作成されたか、ユーザ・モード入出力オペレーションに応答してＮＴエグゼクティブによって作成されたかとは無関係である）。

【０１５２】ソース処理コンポーネント２３８のインスタンス（ファイル・オブジェクトで表されている）は出力ピン・インスタンス２４０（これもファイル・オブジェクトで表されている）が関連づけられており、これは別のフィルタ・インスタントの接続のソースとなっている。変換フィルタ２４４の「子」である入力ピン・インスタンス２４２は詳しく前述したように、出力ピン・インスタンス２４０への参照をもつように作成されている。同様に、入力ピン・インスタンス２４８をもつシンク・フィルタ２４６は出力ピン・インスタンス２５０に接続され、これは変換処理コンポーネント２４４と関係づけられている。

【０１５３】相互接続されたカーネル・モード・ソフトウェア・ドライバのシステムでは、バッファ・アロケータ・メカニズムは入力ピン・インスタンスと関係づけられ、入力ピン・インスタンス上に作成または形成されると言われている。さらに、出力ピン・インスタンスは必要ならば、バッファ・アロケータ・メカニズムへの参照を論理的にもち、出力ピン・インスタンスをもつフィルタはこの参照を利用してバッファ・フレームの割当てと新しいフレームへのデータ転送を行ってから、コントロールを別の処理コンポーネントへ引き渡す。すでに説明したように、ヌル参照は、新しいフレームへのデータ転送が必要でないこと、および、処理を既存フレームで行うことができることを示している（つまり、処理の後、データは同じバッファ・フレームに戻される）。バッファ・アロケータ参照が存在するかどうかは、フィルタ・グラフを作成したサード・パーティ制御エージェントの初期交渉によって判断される。

【０１５４】入力ピン・インスタンス２４８上に形成されたバッファ・アロケータ・メカニズムはファイル・オブジェクトで表されているが、破線２５４は、出力ピン・インスタンス２４０がバッファ・アロケータ２５２を表すファイル・オブジェクトへの参照（例えば、ポインタまたはハンドル）をもっていることを示している。図１５に示す例では、メモリのフレームはシステム・メモリ２５６から割当てられている。

【０１５５】フィルタは種々の方法で相互接続できるので、バッファ・アロケータは、用意されている場合であっても、必要でないことがある。バッファ・アロケータ

のインスタンスを表すファイル・オブジェクトが作成されるのは、フィルタを相互接続するサード・パーティ制御エージェントが必要であると判断した時だけである。このようにすると、フィルタはさまざまな構成で柔軟に接続でき、かつ、最適なデータ転送特性はそのまま保持される。さらに、デフォルトのシステム・バッファ・アロケータを用意すると、ドライバ開発者の開発作業をさらに削減することができる。

【０１５６】サード・パーティ制御エージェントは、架空上のモデルを構築する一部として接続ピンに必要なバッファ要件についても照会してから、実際のフィルタ接続を行う。いくつかの実装では、ピンのインスタンス生成前に照会を行うことが可能になっているが、本実施の形態では、必要なバッファ要件を確かめるためには、その前に実際の接続ピン・インスタンスを作成しておくことが要求される。さらに、本明細書に開示されている実施の形態では、照会は前述した集合メカニズムの使用を通して行われる。

【０１５７】サード・パーティ・クライアントまたは制御エージェントはカーネル・モード・フィルタの相互接続を完成してフィルタ・グラフを作成したとき、次に、入力ピン・インスタンス（またはシンク・ピン・インスタンス）のハンドルを使用してアロケータ要件の交渉を開始する。規定により、入力ピン・インスタンスは必要なバッファ割当て量を定義し、バッファ割当てメカニズムを提供するのに対し、出力ピン・インスタンスは入力ピン・インスタンスに関連する適切なバッファ割当てメカニズムへの参照をもっている。この分野の当業者ならば理解されるように、他の規定を使用すると、バッファ割当ての役割を入力ピン・インスタンスと出力ピン・インスタンスの間で逆にするといったように、同じ結果を効果的に達成することができる。

【０１５８】バッファ割当ての要件は、すべてのコンプライアント・フィルタによってサポートされる特定のプロパティ集合メカニズムを使用して確かめられる。理解されるように、「バッファ・アロケータ」のプロパティ集合は、他の集合と同じようにさまざまな方法で構成することが可能である。例えば、プロパティ集合は単一のプロパティをもつことが可能であり、その場合、そのプロパティはセグメント化された情報をもつデータ構造である。あるいは、プロパティ集合は複数のプロパティをもつことも可能であり、その場合、１つ１つは異なるフレーミング要件エレメント用となっている。データ構造からなる単一プロパティは、すべてのフレーミング要件情報を取り出すためにサード・パーティ制御エージェントが必要とするプロパティ集合照会が少なくなるので、ある状況によってはより効率的である。さらに、単一データ構造は要件情報を照会するためだけでなく、実際のバッファ・アロケータの作成時にパラメータを指定するためにも使用できる。

55 56

【０１５９】以下の表はデータ構造に、または個別的プロパティとして組み入れることができるフレーミング要件エレメントの非排他的リストを示している。また、これらの表には、このようなエレメントが実施の形態でど＊のような使い方がされるかの解説も含まれている。

【０１６０】

【表７】

（表３）

バッファ・アロケータ・フレーミング要件エレメント	
エレメント	説明
制御オプション（作成）	このエレメントは特定の接続ピン・インスタンスにおけるバッファ・アロケータの作成時に指定される制御オプションを収めている。オプションには、次のものがある。 システム・メモリ：この制御オプションはバッファ・フレーム割当てのためにシステム・メモリを使用することを指定する。これを指定するときは、バッファ・アロケータはシステム・メモリに置かれているプール（割当てプール・タイプ・エレメントに指定されている）からメモリを割当てられなければならない。この制御オプションの指定がないときは、入力接続ピン・インスタンス（またはシンク・ピン）は、システムに接続されたデバイス上のオンボードＲＡＭまたは他の形態の記憶装置に対応づけられているシステム・アドレスを提供する。デバイスとしては、アッドイン・カード、このようなPCMCIAカードなどがある。 ダウン・ストリーム互換性：この制御オプションは作成されるバッファ・アロケータのアロケータ・フレーミング・エレメントがダウン・ストリーム・アロケータと互換性があることを指定する。このオプションは、同所修飾子がコピー・バッファ用のアロケータに割り当てられるとき指定されるのが通常である。フィルタが特定のフレームを変更する必要がなければ、ダウン・ストリーム・アロケータから追加フレームを割当てることなくそのフレームをダウン・ストリーム・フィルタに渡すことができる。

【０１６１】

【表８】

（表３のつづき）

要件（照会）	このエレメントは接続ポイントの照会時に戻される接続ピン・インスタンス（入力またはシンク）のアロケータ要件を収めている。要件としては次のものがある。 同所：これは、フィルタが同所変更を実行できることを接続ピン・インスタンスが示していたことを示している。そうでない場合は、バッファは変更データを受け入れるように割当てられていなければならない。 システム・メモリ：接続ピン・インスタンス（入力またはシンク）はすべてのバッファ・フレーム割当てのためのシステム・メモリを必要とする。この要件が設定されていないと、入力接続ピン・インスタンス（またはシンク・ピン）は物理デバイス上のオンボードＲＡＭまたは他の形態の記憶装置に対応づけているシステム・メモリを提供する。 フレーム保全性：この要件はダウンストリーム・フィルタが特定フレームのデータ保全性を維持することを接続ピン・インスタンスが要求していることを示している。これは、現行フレームを正しく処理するために先行フレームの統計が必要であるとき使用できる。 必須の割当て：この要件は接続ポイントが送信されるフレームを割当てらなければならないことを要求している。 優先専用フラグ：この要件フラグは通知される他の要件が優先専用であることを示している。つまり、これらの要件は優先オペレーション方法を示していることを意味するが、接続ポイントはそれでも正しく働き、特定の要件に合致しないフレームを処理する。
割当てプール・タイプ	このエレメントはカーネル・モード・エンティティがバッファ・フレーム割当てのためのシステム・メモリをどの割当てプール・タイプから受け取るか判断する。

【０１６２】

【表９】

（表３のつづき）

未処理フレーム	このエレメントは割当てることができる許容未処理バッファ・フレームの総数を示している。このように、バッファの総サイズを制御できる。さらに、このエレメントにゼロを指定することは、フィルタ（特定の接続ピン・インスタンスを通して）は未処理フレーム数を制限する必要がないことを意味する。
フレーム・サイズ	このエレメントはバッファ・フレームの総サイズを指定する。この場合も、このエレメントにゼロを指定することは、フィルタ（特定の接続ピン・インスタンスを通して）は未処理フレーム数を制限する必要がないことを意味する。
ファイル位置合わせ	このエレメントはフレームをある境界上に割当てるときの位置合わせを指定する。例えば、８進バイト位置合わせが指定できる。さらに、実施の形態では、最低位置合わせ機能は４倍長境界上で行なわれるので、現存ＰＣハードウェア・アーキテクチャでのデータ・アクセス速度が最適化されている。

【０１６３】この分野の当業者ならば理解されるように、組み入れることができる関係フレーミング・プロパティはほかにもある。さらに、ここで説明しているバッファ割当てメカニズムは、表３に指定されているものより多いバッファ・フレーム・エレメントが組み込まれているか、そのサブセットが実装されているかに関係なく、ほぼ同じように機能する。

【０１６４】フィルタ・グラフ要件がサード・パーティ制御エージェントによって判断されると、次に、適切なカーネル・モード・バッファ・アロケータを適切な入力ピン・インスタンス上に作成することができる。クライアントは適切な接続ピン・インスタンスのハンドルを使用し、バッファ・アロケータの適切な作成パラメータを指定することによってバッファ・アロケータ・インスタンスを作成する。このようにして得られた、バッファ・アロケータを表すファイル・オブジェクトは接続ピン・インスタンスの子となり、そのファイル・オブジェクトとフィルタ自体のインスタンスを表すファイル・オブジェクトからのコンテキスト情報を使用してそれを正しく作成する。

【０１６５】言い換えれば、接続ピン・インスタンスの作成をバリデーションし、メッセージを特定の接続ピン・インスタンスの適切なハンドラへ転送するための、前述したものと同じメカニズムはバッファ・アロケータのインスタンスの作成にも同じように適用される。この場合も、NtCreateFileコールはサード・パーティ制御エージェントが使用できるＡＰＩの一部としてハイレベル関数コールの中でラップされることになる。

【０１６６】バッファ・アロケータ・インスタンスはここで説明している実施の形態では、作成されるとすれば、入力ピン・インスタンスだけで作成される。

【０１６７】バッファ・アロケータ・インスタンスが適切なＡＰＩを通して出力ピン・インスタンス・ハンドルに対して指定されていなければ、フィルタは、ストリーム入出力コントロールを通して渡されたストリーム・セグメントがフィルタの要件に合致しているものと想定できるので、データを同所で自由に変更することができる。

【０１６８】システムに用意されているデフォルト・アロケータをフィルタ開発者によって使用すると、入力接続ピン・インスタンスにバッファ割当て機能をもたせる作業が単純化される。このデフォルト・アロケータはデータをシステム・メモリから転送することができるデバイス・ドライバのためにシステム・メモリ・バッファ・フレーム割当てを行うが、特定のメモリ割当てプロパティを必要とする。デフォルト・バッファ・アロケータを使用すると、フィルタ開発者はバッファ割当てを行うコードを実際に準備する作業から解放される。しかし、その場合でも、フィルタは適切なプロパティ集合をサポートすることによって必要バッファ割当て量要求をサポートするように書かれることになる。

【０１６９】デフォルト・アロケータを使用するためには、フィルタ設計者は、（１）必要バッファ割当て要件要求に応えるコードを準備し、（２）デフォルト・アロケータ作成ハンドラ参照を、デフォルト・アロケータが関係する特定接続ピン・インスタンスのバリデーション・テーブルに入れる。言い換えれば、アロケータ・タイプ・メカニズムの作成要求がフィルタを通して渡されると、特定のＧＵＩＤストリングはバリデーション・テーブルで突き合わされ、デフォルト・アロケータ作成ハン

ドラへのアクセスができるようにする。

【０１７０】デフォルト・バッファ・アロケータはシステム・メモリを使用し、作成要求の一部として渡されたバッファ・アロケータ・フレーミング・プロパティに従って動作する。この種のデフォルト・アロケータはその特定機能と相互接続順序に応じて、種々の変換フィルタで使用される可能性がある。

【０１７１】オンボード・メモリまたは他のデバイスに依存する記憶方式のためのバッファ・アロケータを要求するフィルタは、バッファ・アロケータ・プロパティ集合とメソッド集合をサポートすることで専用のバッファ・アロケータを用意することができる。フィルタ専用アロケータの場合、フィルタは機能全体を実装するプログラム・コードを準備することに責任を有している。しかし、バッファ・アロケータのオペレーションは、デフォルトであるか、フィルタ専用であるかに関係なく、どのバッファ・アロケータの場合も同じであるので、サード・パーティ・エージェントはフィルタとバッファ・アロケータを正しく相互接続することができる。

【０１７２】バッファ・アロケータを表すファイル・オブジェクトは、正常に作成されたときは、データ構造を指すポイントをファイル・コンテキスト・エリアに置いている。このデータ構造には、ＩＲＰを種々のＩＲＰコード（例えば、作成、入出力コントロールなど）に基づいて指定のハンドラへ送るためのディスパッチ・テーブルが、バッファ・アロケータの状態を保持するための他のデータ・エリアと構造と共に収められている。トラッキングできる実装依存情報のいくつかを挙げると、バッファが関係する接続ピン・インスタンスのファイル・オブジェクトへの参照、割当てフレーミング要件データ構造への参照、イベントを待っているクライアントのイベント・キュー、未処理になっている割当て要求（例えば、ＩＲＰによって受信されたもの）のキューなどがある。

【０１７３】ここで開示されている実施の形態のバッファ割当てメカニズムが利用できるインタフェース、つまり、通信方法は２つある。まず、すべてのアロケータはユーザ・モード・クライアントと正しく通信するためにはＩＲＰベースのインタフェースを提供しなければならない。オプションとして、割当てプール・タイプがオペレーティング・システムのディスパッチ・レベル（サービスの限定されたサブセットが利用できるが、低優先度のタスクがそのプロセッサでブロック・アウトされるような高さの優先度レベル）でサービスを受けることができる場合は、関数テーブル・インタフェースがサポートされるので、相互接続されたフィルタは直接関数コールを使用して（ＤＰＣ処理時に）パフォーマンスを向上することができる。これにより、ＩＲＰを入出力マネージャを通して受け渡すという余分のオーバヘッドを発生することなく、あるいは実行スレッドをスケジュールしてコンテキスト切替えを待つことなく、イベント通知を伝達することができる。

【０１７４】ＩＲＰベースのインタフェースはＩＲＰを次のように連続的に処理する。割当て要求が渡されると、バッファ・アロケータはその要求を完了し、割当てられたバッファ・フレームを指すポインタを戻し、すべてのフレームが割当てられていれば、アロケータはＩＲＰに保留(pending) のマークを付け、ＩＲＰをアロケータの要求キューに追加し、ほぼＦＩＦＯ順に処理される。最後に、アロケータは保留中というステータスをコール側に戻す。

【０１７５】バッファ・フレームがアロケータに利用可能になったとき、アロケータはＩＲＰベースのインタフェースの要求キューに置かれている最初の要求を完了することを試みる。以前にサービスを受けることができなかったＩＲＰはこの要求キューに置かれ、処理のために新たに解放されたフレームを待つことになる。これとは逆に、要求キューで待たされている作業がなければ、そのフレームは空きリストに戻される。

【０１７６】直接関数コール・テーブルを使用するディスパッチ・レベル・インタフェースは次のように動作する。割当て要求が関数コール（これはＤＰＣ時に行うことができる）によって渡されると、アロケータは使用可能なフレームがあれば、そのフレームを指すポインタを戻し、さもなければ、ヌル（空白）を直ちに戻す。この場合、カーネル・モード・リクエスタは空きフレームがあることを知らせる空きイベント通知を待つことができる。この通知を受けると、カーネル・モード・リクエスタは割当て要求を再度試みる。

【０１７７】なお、ディスパッチ・レベル・インタフェースとＩＲＰベースのインタフェースのどちらもが使用可能な空きフレームを奪い合うことが起こり得る。また、完了待ちに置かれている割当て要求ＩＲＰが要求キューに残っていれば、アロケータは、コール側がフレームを解放したとき現ＩＲＱＬが受動レベルになければ、ワーカ・アイテム(worker item) をスケジュールしなければならないが、これは、直接コール・インタフェースを使用することはＤＰＣレベルにある可能性を意味するからである。基本的には、ＩＲＰの待ち行列はワーカ・アイテムが実行されるまでは空きフレームを探し出さない。

【０１７８】さらに、ここで説明しているバッファ・アロケータ・メカニズムはＭＤＬ（memory descriptor list：メモリ記述子リスト）と一緒に使用するのに適しているので、バッファ間の転送数をさらに削減することができる。このような実装によると、ＮＴオペレーティング・システムのシステム・メモリ機能とのシームレスな相互作用が向上することになる。

【０１７９】次に、図１６を参照して説明すると、同図には前述したバッファ割当てメカニズムを利用する図１

１と図１２の相互接続フィルタ・システムが示されている。制御エージェント１７０はフィルタ間の相互接続を行ったあと、必要バッファ量について各入力ピン・インスタンスに照会する。図に示すように、サウンド・レンダリング・フィルタ１７８はバッファ・フレームをサウンド・カード・メモリ２５８から割当てる必要があり、デコンプレッサ・フィルタ１７４はデータをディスク・ドライブ２６２から直接に受け取ることができるシステム・メモリ２６０からメモリを割当てることになる。

【０１８０】制御エージェント１７０はファイル・オブジェクトで表され、入力ピン・インスタンス１８８上に形成されるバッファ・アロケータ２６４を作成する。バッファ・アロケータ２６４はバッファ・フレームをサウンド・カード・メモリ２５８から割当て、バッファ・アロケータ２６４への参照は破線２６６で示された出力ピン・インスタンス２０４に設定される。この参照はバッファ・アロケータ２６４を表すファイル・オブジェクトへのハンドルとなり、必要時にバッファ・フレームを割当てるためにデコンプレッサ・フィルタ１７４によって使用されてからデータが新しいバッファに転送される。

【０１８１】同じように、バッファ・アロケータ２６８もファイル・オブジェクトによって表され、入力ピン・インスタンス２０６上に作成される。このバッファ・アロケータ２６８はシステム・メモリ２６０の割当てを管理する。制御エージェント１７０はバッファ・アロケータ２６８を作成したあと、破線２７０で示すようにその参照を出力ピン・インスタンスにストアする。この場合も、バッファ・アロケータ２６８はシステム・メモリ２６０間のバッファ・フレームの割当てを担当し、データがディスク２６２からそこに転送できるようにする。

【０１８２】制御エージェントは出力ピン・インスタンス１９２のバッファ・アロケータ参照の値にヌル（空白）を入れ、同所変換が行われることを示し、効果フィルタ１７６はサウンド・カード・メモリ２５８内の既存バッファからデータを読み取り、必要とされる変換または効果を適用したあとでデータをサウンド・カード・メモリ２５８に戻す。逆に、制御エージェントがバッファ・アロケータ参照を設定していなければ、値が空白であり、同所変換が行われると想定される。

【０１８３】次に、図１７のフローチャートと図１６のロジック図を参照して、バッファ・アロケータのオペレーションについて説明する。このプロセスは相互接続が行われたあとで実行され、ストリーム読取りと書込みは制御エージェント１７０からリーダ・フィルタ１７２に渡される。

【０１８４】最初に、リーダ・フィルタ１７２はステップ２７２で、デコンプレッサ・フィルタ１７４のバッファ・アロケータ２６８を使用してシステム・メモリにフレームを割当てる。リーダ・フィルタ１７２の出力ピン・インスタンス２０８はライン２７０で示すように、バッファ・アロケータ２６８を表すファイル・オブジェクトへのハンドルを受け取るので、直接にアクセスしてバッファ・アロケータ２６８を制御することができる。

【０１８５】ファイル・リーダ・フィルタ１７２はシステム・メモリ２６０内の実際のバッファ・フレームへのアクセス権を得ると、ステップ２７６で矢印２７４で示すように、ディスク２６２からのデータをフレームに入れる。このことから理解されるように、リーダ・フィルタ１７２はデータをシステム・メモリ２６０に持ち込むとき、変換または他の操作をデータに対して行うことができる。

【０１８６】次に、ファイル・リーダ・フィルタ１７２はステップ２７８でデコンプレッサ・フィルタ１７４へのストリーム書込みを開始する。このストリーム書込みはＩＲＰによってＮＴ入出力マネージャに渡される。ステップ２８０で、デコンプレッサ・フィルタ１７４はバッファ・アロケータ２６４を使用してサウンド・カード・メモリ２５８のフレームを割当てる。デコンプレッサ・フィルタ１７４がバッファ・アロケータ２６４を知っているのは、そこへのハンドルが出力ピン・インスタンス２０４に対してストアされていたためである。

【０１８７】デコンプレッサ・フィルタ１７４はデータを伸張し、矢印２８４で示すようにそのデータを、サウンド・カード・メモリ２５８に以前に割当てられたフレームに転送する。なお、データを伸張するとき、サウンド・カード・メモリから割当てられたフレームがシステム・メモリに存在するよりも多くなっている場合がある。この余剰バッファ・フレーム比が必要になるのは、データの伸張効果を受け入れるためである。

【０１８８】重要なことは、データを一方のバッファから他方のバッファへ転送するとき、転送されるデータ量が１：１の対応関係でない場合があることである。言い換えれば、受信側バッファが必要とするスペース（またはフレーム数）は、バッファ転送間でどのようなフィルタリングまたは変換が行われるかに応じて、多くなる場合と少なくなる場合とがある。

【０１８９】デコンプレッサ・フィルタ１７４は特定フレームの伸張を終えると、ＮＴ入出力マネージャの機能を使用してストリーム書込みを効果フィルタ１７６に渡す。効果フィルタ１７６はステップ２８８でストリーム書込みＩＲＰを受け取り、既存サウンド・カード・メモリ２５８でデータを同所処理する。この効果処理はデータを１：１で置換するのと同じであるので、必要なバッファ・メモリは多くなることも、少なくなることもない。

【０１９０】効果フィルタ１７６がデータの同所処理を終えると、ストリーム書込みＩＲＰはステップ２９０でサウンド・レンダリング・フィルタに渡される。この場合も、ストリーム書込みＩＲＰをサウンド・レンダリング・フィルタ１７８に転送させるメカニズムは効果フィ

ルタ１７６によってコールされたＮＴ入出力マネージャの機能である。

【０１９１】最後に、サウンド・レンダリング・フィルタ１７８はステップ２９２でストリーム書込みＩＲＰを受け取り、サウンド・カード・メモリ２５８に存在するサウンド・データの実際のレンダリングを形成するようにサウンド・カード・ハードウェアを制御する。この時点で、以前に割当てられていたサウンド・カード・バッファ・フレームは書込み要求を満たすために再使用することも、未処理の要求がなければ解放することもできる。バッファ・フレームが使用可能であることはデコンプレッサ・フィルタ１７４に知らされるので、待ちに置かれているストリーム書込みを使用してデータを処理し、割当てが解放されたバッファ・フレームに入れることができる。同様に、システム・メモリ２６０のバッファ・フレームは再使用されるか、解放される。

【０１９２】次に、図１８を参照して説明すると、図はシステムの概要を示す図であり、そこでは、リアルタイムで生成され、「ライブ」であると呼ばれる２つのデータ・ストリームをマスタ・クロック・メカニズムを使用して同期化することが可能になっている。オーディオ・レンダラ２９６は矢印３００で示すように、ライブ・オーディオ・ストリーム２９８を受信して処理する。オーディオ・レンダラ２９６がデータを処理するとき、オーディオ・レンダラ２９６がデータ・サンプルを適切な電子信号に変換して、この信号を矢印３０４で示すようにスピーカ３０２に送信すると、データはスピーカ３０２からサウンドとして知覚される。

【０１９３】同様に、ビデオ・レンダラ３０６は矢印３１０で示すようにライブ・ビデオ・ストリーム３０８を受信して処理する。ビデオ・レンダラ３０６によって処理されると、データ・サンプルは適切なビデオ信号に変換され、矢印３１４で示すようにモニタ３１２に送られ、そこでデータ・サンプルはイメージとして知覚される。

【０１９４】代表例として、ライブ・ビデオ・ストリーム３０８とライブ・オーディオ・ストリーム２９８は同時にレンダリング・コンポーネントに着信する。コヒーレントなプレゼンテーションを再現するために、スピーカ３０２とモニタ３１２でレンダリングされるときビデオをオーディオと整合させることは重要であるので、ライブ・ビデオ・ストリーム３０８とライブ・オーディオ・ストリーム２９８の処理は同期化されなければならない。

【０１９５】これを行なうためには、一方のデータ・ストリームがマスタとなるように選択される。ここでは、説明の便宜上、ライブ・オーディオ・ストリーム２９８がマスタ・データ・ストリームとなるように任意的に選択されている。なお、ライブ・ビデオ・ストリーム３０８がマスタとして選択されることも、他のデータ・ストリームがマスタとして選択されることも可能であり、それを妨げるものはなにもない。さらに、フィルタ・グラフの外部にあるデータ・ストリームに基づく外部クロックを使用して、ライブ・オーディオ・ストリーム２９８とライブ・ビデオ・ストリーム３０８を共に同期化することが可能である。従って、図示のマスタ・クロック・メカニズム３１６では、実線でオーディオ・レンダラ２９６と結ばれ、破線３１８でビデオ・レンダラと結ばれているが、このことは、ビデオ・レンダラがオーディオ・ストリームをマスタとして使用しているマスタ・クロック３１６から同期化情報をなんらかの方法で受信することを示している。

【０１９６】マスタ・クロック３１６は、オーディオ・レンダラ２９６での処理レートをオリジナル・オーディオ・ストリーム２９８のレートと整合するためにオーディオ・レンダラ２９６によって使用することも可能である。言い換えれば、ライブ・オーディオ・ストリーム２９８は、別のシステム上のディジタル化フィルタ／ディジタル化ハードウェアなどの別の処理コンポーネントによって処理されている。オーディオ・ストリームはディジタル化サンプルから構成されているので、これらのサンプルはオリジナル処理コンポーネントのハードウェアおよびオシレータに従うレートで作られている。オリジナル処理コンポーネントがライブ・オーディオ・ストリーム２９８を出力するときの、この発生レート(origination rate)はライブ・オーディオ・ストリーム２９８のインターバル情報の中に入っているか、そこから得られている。

【０１９７】データ・ストリームに入っているサンプル情報には、いくつかの方法で時間インターバル情報を収めることができる。１つの方法は、各サンプルまたはサンプル・グループにタイムスタンプを関連づけることである。さらに、データのフォーマット自体にサンプル・レートを意味させることもできる。例えば、データ・フォーマットの定義によって、一定時間インターバル（例えば、ビデオ・サンプルでは１／３０秒単位、オーディオ・データでは１／２２，０５０秒単位）で各サンプルをとることができる。さらに、タイムスタンプがレンダリング処理ユニットでクロックと整合されるプレゼンテーション時間を示すようにすれば、特定のサンプルをいつレンダリングまたは「プレゼンテーション」するべきかを判断することができる。また、タイムスタンプは、タイムスタンプ間の相対的差から時間インターバル情報を知ることができる、ある種の他の時間値を示すこともできる。最後に、データ・フォーマットとタイムスタンプ情報をミックスして使用すると、処理される参照データ・ストリーム内の位置に基づいてタイムクロックを作成するときに必要になる時間インターバル情報を得ることができる。

【０１９８】以上から理解されるように、時間またはデ

ータ・ギャップに基づいて、データ・ストリームの中に「中断(discontinuities)」をコーディングすることができる。時間的中断とは、レコーディングがある時刻に終了し、その後のある時刻に突然開始するときのように、ストリームが単純に別のプレゼンテーション時刻にジャンプすることである。データの中断とは、沈黙の周期をあるコードで表し、そのあとに沈黙の持続時間が続くといったように、ストリームがそのストリームの中により効率的にコーディングできる反復データの周期をもつことができる場所である。

【０１９９】マスタ・クロック３１６から得られるある時間値はデータ・サンプルのストリームに入っている時間インターバル情報に基づいており、メディア時間と呼ばれている。メディア時間は、マスタ・ストリーム内の位置を表しているので位置時間であり、この場合、マスタ・ストリームはライブ・オーディオ・ストリーム２９８として選択されている。さらに、処理がオーディオ・レンダラ２９６で終了したとき、メディア時間もマスタ・クロック３１６で凍結される。データ・ストリーム内の中断はストリーム内のその「位置」まで処理され、基礎となるハードウェア・オシレータまたは他のソースを使用して時間の進みをトラッキングすることによって、中断にある間進み続ける。このようにすると、複数のデータ・ストリームを、時間またはデータの中断を有するデータ・ストリームと同期させることができる。

【０２００】ビデオ・レンダラ３０６をマスタ・クロック３１６に入っているメディア時間と同期させ、マスタ・クロックの方をライブ・オーディオ・ストリーム２９８に入っている時間インターバル情報に基づかせると、メディア時間はライブ・オーディオ・ストリーム２９６がオーディオ・レンダラ２９６で処理される時だけ進む（「刻時(ticks)」）するので、停止または休止状態を管理するためのオーバヘッドが不要になる。

【０２０１】以上から理解されるように、ビデオ・ストリーム３０８も、同期化と同時にレート整合を行う必要がある。ビデオ・ストリーム３０８のレート整合はライブ・オーディオ・ストリーム２９８およびオーディオ・レンダラ２９６に関して以下で説明するのと同じ方法で行われる。当然のことであるが、ライブ・ビデオ・ストリーム３０８のレート整合はビデオ・レンダラ２９６のハードウェア・オシレータ（物理時間）に基づいている。

【０２０２】マスタ・クロック３１６から得られるものとして、オーディオ・レンダラ２９６内の基礎となるハードウェアに基づく物理時間もある。物理時間は、オーディオ・レンダラ２９６がアクティブ状態でオーディオ・データを処理しているか、停止または休止状態にあるかに関係なく、進んでいくのが通常である。さらに、メディア時間（発生側ハードウェア処理レートを表す）および物理時間（オーディオ・レンダラ２９６での実際の処理レートを表す）の進みレートをサンプリングすると、レートが相対的に整合されるようにオーディオ・レンダラで調整を行ってライブ・オーディオ・データ２９８を処理することができる。

【０２０３】レート整合は、ストリーム・データをローカル・ディスクや他の記憶デバイスから読み取るといったように、データ発生のレートがプロセッサの制御下に置かれているときは重要な問題とならないが、データを連続的に発生する「ライブ」データ・ストリームを処理し、発生レートがプロセッサの制御下に置かれていないときは重要な問題となる。ここで用いられている「ライブ」データまたはメディア・ストリームとは、処理コンポーネントの制御下に置かれていないレートで発生するストリームのことであり、完全にリアルタイムで処理されないと失われるようなストリームである。ライブ・データ・ストリームの例としては、次のようなものがあるが、これらに限定されるものではない。つまり、ストリームがローカルまたはネットワーク接続を介してリアルタイムで発生するようなリアルタイム・フィード、リアルタイム・フィードであるか、ネットワークを介して送られてきたストア済みデータであるかに関係なく、ストリームをネットワーク接続から受信することなどがある。いずれの場合も、ストリームはその受信と同時に処理されなければならない。

【０２０４】ライブ・メディア・ストリームが発生するデータが多すぎるため、オーディオ・レンダラ２９６が処理できないときは、オーディオ・レンダラ２９６は、バッファリング容量に限りがあるためデータ「あふれ(folding)」と呼ばれる状態が起こり、最終的にはデータを失うことになる。他方、ライブ・オーディオ・ストリームがオーディオ・レンダラ２９６よりも低レートでデータを発生する場合には、オーディオ・レンダラには、処理すべきデータがなくなるというデータ「枯渇(starvation)」状態が起こることになる。どちらの場合も、オーディオ・レンダラ２９６でのレンダリング・レートを、ライブ・オーディオ・ストリーム２９８内の時間インタバール情報で表されている発生レートに一致するように調整を行うと、ライブ・オーディオ・ストリーム２９８のレンダリングが高品質に行われるという利点がある。

【０２０５】レート補正調整を行う１つの方法は、オーディオ・レンダラにデータ枯渇が起こらないようにデータ・サンプルを周期的に複製するか、あるいはオーディオ・レンダラがその能力を超えて出力するライブ・オーディオ・ストリーム２９８に追いつくようにサンプルを捨てることである。どちらの場合も、データ・サンプルの複製または破棄は、スピーカ３０２で知覚される影響量が最小になるようにストリーム処理時に間隔を置いて行うのが賢明である。

【０２０６】マスタ・クロック３１６と直接に同期をと

るのではなく、マスタ・クロック３１６に入っているメディア時間に一致するように別のクロックのタイムベースを変換するか、あるいはマスタ・クロック３１６のメディア時間に一致するように別のメディア・ストリーム内のプレゼンテーション時間情報を変換するようにすると便利である。この変換を行いやすくするために、マスタ・クロック３１６は２つの別個の時間値、つまり、メディア時間値と他のコンポーネントに共通の参照時間値とからなる相関メディア時間値を提供し、これをアトミック・オペレーションで行ってタイミング誤差の発生を低減化している。相関物理時間値も提供される。

【０２０７】参照時間は、すべての処理コンポーネントが利用できるＰＣクロックまたはシステム上の他のクロック・メカニズムに基づいているのが代表的であるが、別のクロックを選択することも可能である。アトミック・オペレーションはネーティブ・システム機能を利用して、中断または干渉量を最小にして両方の値が取り出せるように実装する。相関時間（メディアまたは物理）はフィルタなどの別の処理コンポーネントが使用して、メディア時間と参照時間との相対的オフセットまたはデルタを判断し、発生誤差を最小にして必要な変換を行うことができる。これについては、以下で詳しく説明する。

【０２０８】次に、図１９および図２０を参照して説明すると、上述したクロック・メカニズムは前述した相互接続フィルタ・システムと共に実装されている。図１９、オーディオ・ストリームと同期がとられ、マスタ・クロック・メカニズムから同期化情報を受け取るライブ・ビデオ・ストリーム処理フィルタの集合を示している。図２０は、マスタ・クロック・メカニズムのマスタ時間参照を生成するために使用されるライブ・オーディオ・ストリームをレンダリングするための相互接続フィルタの集合を示している。

【０２０９】まず、図１９に示すビデオ・ストリーム処理コンポーネントについて説明すると、ライブ・ビデオ・ストリーム３２０は矢印３２４で示すように、ビデオ・リーダ・フィルタ３２２によって受信される。ライブ・ビデオ・ストリーム３２０のソースとしては、コンピュータに置かれたオンボード・ディジタル化ハードウェア、コンピュータ・ネットワーク経由で受信されるディジタル・サンプルのバケットなどがある。ライブ・ビデオ・ストリーム３２０は実際のビデオ・イメージにレンダリングできるディジタル化サンプルから構成され、これらのサンプルは前述したように時間インターバル情報が関連づけられている。この場合も、ここで説明している相互接続フィルタ・システムによれば、ビデオ・リーダ・フィルタ３２２はファイル：オブジェクトで表されている。

【０２１０】ビデオ・リーダ・フィルタ３２２は、ビデオ・デコンプレッサ・フィルタ３３０上の入力ピン・インスタンス３２８に接続された出力ピン・インスタンス３２６をもっている。ビデオ・デコンプレッサ・フィルタ３３０はライブ・ビデオ・データを圧縮フォーマットで受信し、そのデータをビデオ・レンダリング・ハードウェアに受け付けられるフォーマットに伸張する。ビデオ・データのフレームを処理すると、デコンプレッサ・フィルタはコントロールをビデオ・レンダリング・フィルタ３３２に渡す。ビデオ・レンダラ・フィルタ３３２とビデオ・デコンプレッサ・フィルタ３３０との接続は、それぞれ出力ピン・インスタンス３３４と入力ピン・インスタンス３３６によって行われる。

【０２１１】入力ピン・インスタンス３３６は同期化のためのタイミング指示を、ボックス３３８と破線３４０で示すように、オーディオ・ストリーム処理コンポーネント内のマスタ・クロック・メカニズムから受信する（図１８参照）。この同期化タイミング情報を受信するには、さまざまな方法が可能である。まず、図１９と図２０に示すフィルタ・グラフ・トポロジを相互接続する制御エージェント３４２はマスタ・オーディオ・ストリーム内の特定の位置（つまり、時間）に基づいてクロック・メカニズム３４４にイベント通知を行わせることも、特定の時間インターバルを選択してインターバル・イベント通知を行うことも可能である。別の方法として、ビデオ・レンダリング・フィルタ３３２はクロック・メカニズム３４４に照会することも可能である（図２０参照）。クロック・メカニズム３４４はファイル・オブジェクトで表されているので、制御エージェント３４２はクロック・メカニズム３４４への該当する参照を入力ピン・インスタンス３３６またはビデオ・レンダラ・フィルタ３３２に対して行うことができるので、ビデオ・レンダラ・フィルタ３３２はクロック・メカニズムに照会して相関時間値を得て、必要ならば変換を行うことも、メディア時間値を得て処理が同期化されていることを確かめることもできる。

【０２１２】最後に、ビデオ・レンダラ・フィルタ３３２はビデオ・ハードウェア３４６を制御し、データを矢印３４８で示すようにそこへ転送する。ビデオ・ハードウェアはデータをレンダリングし、イメージをモニタ３５０から表示させるビデオ信号を生成する。

【０２１３】次に、図２０を参照して、ライブ・オーディオ・ストリームの処理コンポーネントについて詳しく説明する。この場合も、同じ制御エージェント３４２は、必要ならば入力ピン・インスタンスおよび出力ピン・インスタンスと一緒にそれぞれのフィルタ・インスタンスを作成し、それぞれのフィルタ間の相互接続を行うが、カーネル・モード・フィルタ・グラフ・トポロジ全体を作成するのは制御エージェントである。

【０２１４】ライブ・オーディオ・ストリーム３５２は最初に、矢印３５６で示すようにオーディオ・リーダ・フィルタ３５４によってフィルタ・グラフ・トポロジに持ち込まれる。データはそれぞれの出力ピン・インスタ

ンス３６０と入力ピン・インスタンス３６２を使用してオーディオ・リーダ・ファイルに接続されているオーディオ・デコンプレッサ・フィルタ３５８によって伸張される。この場合も、ここで説明している各フィルタと接続ピン・インスタンスはファイル・オブジェクトで表され、前述したように作成される。

【０２１５】オーディオ・デコンプレッサ・フィルタはそれぞれ出力ピン・インスタンス３６６と入力ピン・インスタンス３６８を通してオーディオ・レンダラ・フィルタ３６４に接続されている。入力ピン・インスタンス３６８と関連づけられているものとして、メディア時間、物理時間、および／または相関時間を提供またはその通知を他のエンティティへ送信するクロック・メカニズム３４４のインスタンス、特に、ボックス３７０と破線３７２で示すようにビデオ・レンダラ・フィルタ３３２（図１９参照）の入力ピン・インスタンス３３６がある。クロック・メカニズム３４４の実際の実装と利用可能性については、以下で詳しく説明する。このクロック・メカニズム３４４を使用すると、他のデータ・ストリームは、図２０に示すようにその処理を「マスタ」または「参照」オーディオ・ストリーム３５２の処理と同期化することができる。

【０２１６】オーディオ・レンダラ・フィルタ３６４はオーディオ・ハードウェア３７４を制御し、制御情報とデータを矢印３７６で示すように引き渡す。オーディオ・ハードウェアは、オーディオ・レンダラ・フィルタ３６４の制御の下で、ディジタル化データ・サンプルをスピーカ３７８へ送信し、そこから知覚されるようにするリアルタイム電子信号にレンダリングする。

【０２１７】相互接続カーネル・モード・フィルタを前述したように作成するとき、各フィルタ・インスタンスはシステムが利用できる「デバイス」から作成される。このようにすると、ビデオ・リーダ・フィルタ３２２とオーディオ・リーダ・フィルタ３５４は同じファイル・リーダ・デバイスの別々のインスタンスにすることができる。

【０２１８】さらに、特定のフィルタは接続ピン・インスタンスを作成するために多数の接続ピン・ファクトリをサポートできるので、デコンプレッサ・フィルタ３３０とオーディオ・デコンプレッサは同じデコンプレッサ・デバイスのインスタンスにすることができる。制御エージェントがそれぞれのフィルタ・インスタンス上に接続ピン・インスタンスを作成するとき、フィルタを通して処理されるデータのタイプに応じて異なるピン・ファクトリが接続ピン・インスタンスを作成するときのテンプレートとして選択される。これは、ここで説明している相互接続カーネル・モード・フィルタ・システムが本来的に備えた柔軟性である。

【０２１９】クロック・メカニズム３４４を形成するためには、制御エージェント３４２は、オーディオ・レンダラ・フィルタ３６４上の入力ピン・インスタンス３６８のように、接続ピン・インスタンスに照会してクロック・メカニズムが使用可能であるかどうかを判断するのが代表的である。他の実装では、制御エージェント３４２は接続ピン・インスタンス上にクロック・メカニズムを作成することを試みるだけで、クロック・メカニズムが使用可能かどうかを試行錯誤で判断できるようになっている。さらに、クロック・メカニズムはユーザ・モードで存在し、そのクロックまたは他のメカニズムのカーネル・モード・プロキシを通して通知などを、カーネル・モードのフィルタの入力ピン・インスタンスに送信することができる。

【０２２０】実施の形態では、選択した規則により、タイミング・イベント通知とクロック・メカニズムは入力接続ピン・インスタンスと関連づけられて通常見つけられる。しかし、明らかなように、ファイル・オブジェクトで表されたどのエンティティも、イベント要求をＩＲＰを通して受け取ることも、前のＩＲＰ要求を通して直接プロシージャ・コール・テーブルを検索した後で直接プロシージャ・コールで受け取ることも可能である。

【０２２１】デフォルトとして実装したクロック・メカニズムがシステムに用意されているので、これをソフトウェア・フィルタ開発者に提供または利用できるようにすると、余分のコードを開発しなくてもタイミング機能を得ることが可能である。どのクロック・メカニズム（デフォルト版またはカスタムとして供給）の形成も、作成要求の中でＧＵＩＤストリング値を指定することによって行われるが、このストリング値は接続ピン・インスタンス・バイリデーション・テーブルで使用されて、クロック・メカニズムを接続ピン・インスタンス上に作成するための正しい作成ハンドラをアクセスできるようにする。ファイル・オブジェクトの作成、バリデーション、およびＩＲＰの指定のプロセスは図３～図８を参照して詳しく前述した通りである。接続ピン・インスタンスを作成するのと同じように、クロック・メカニズム作成要求は正しくバス選択し、正当性を調べることができる。

【０２２２】システムに用意されているデフォルト・クロックを使用するためには、ドライバ開発者は作成し、システムに用意している作成ハンドラの中からデフォルト・クロック作成メソッドをコールする。この作成ハンドラは接続ピン・インスタンスを表すファイル・オブジェクトのコンテキスト・エリアから前述したようにアクセスされる。従って、ドライバ開発者はドライバ専用クロック・メカニズムを実装し、適切なドライバ専用作成ハンドラはバリデーション・テーブルに置かれることになる。作成ハンドラ開発の複雑性はクロック機能をコーディングし、実装するのではなく、デフォルト・クロック作成メソッドをコールすることによって低減化される。

73

【０２２３】クロック・メカニズムをこれまでに説明してきた相互接続フィルタ・システムに従って作成する場合は、そのクロック・メカニズムは特定のプロパティ集合とイベント集合をサポートしていなければならない。当然のことながら、デフォルト・クロック・メカニズムは特定の集合をサポートしているが、フィルタ専用クロック・メカニズムの場合は、フィルタ開発者はクロック・メカニズムのプロパティ集合とイベント集合を実装するコードを書いて、用意しておく必要がある。このよう＊

74

＊にすると、クロック・メカニズムへのハンドルをもつ制御エージェントは特定の集合がサポートされるかどうかを照会し、クロック・メカニズムの操作方法を知ることができる。以下に示す表４は、クロック・メカニズムのプロパティ集合に含めることができる、いくつかのプロパティを示したものである。

【０２２４】

【表１０】

（表４）

	クロック・メカニズムのプロパティ
プロパティ	説明
メディア時間	このプロパティは標準単位で表されたクロック上の現メディア時間を戻す。例えば、実施の形態では、これは１００ナノ秒単位で刻時される。クロック・メカニズムが、フィルタまたはフィルタ上の接続ピン・インスタンスのようなタイムスタンプ付きデータまたは制御ストリームを処理する、ある処理コンポーネントによって提示されるような一般的ケースでは、このプロパティによって戻されるメディア時間値はストリーム内の現在位置を反映している。さらに、時間インターバル情報をもつデータまたは制御ストリームはクロックを正しく動作させるために使用できる。基礎となる処理コンポーネントにデータ枯渇状態が起こると、このプロパティによって提示されたメディア時間も停止する。さらに、データ・ストリームまたは制御ストリームがタイムスタンプ付きデータからなり、ストリーム上のタイムスタンプが変更されると、このプロパティによって提示されたメディア時間はこの変更も反映する。従って、タイムスタンプ情報があるフレームから別のフレームに「変換」されると、そのデータ・ストリームに基づくクロック・メカニズムも変換される。最後に、基礎となるコンポーネントがデータまたは制御ストリームの処理を停止または休止するように指示されると、このプロパティによって反映されたメディア時間も停止する。

【０２２５】

【表１１】

（表４のつづき）

物理時間	このプロパティはクロック・メカニズムの現物理時間を戻す。物理時間はある種の基礎となるハードウェア・オシレータに基づく連続走行時間である。デフォルト・クロック・メカニズムの場合は、これはＰＣクロックである。フィルタ専用クロック・メカニズムの場合は、これは実際のレンダリング・ハードウェア・クロックである。物理時間はストリーム枯渇が起こったとき停止しない。また、ストリーム処理が基礎となる処理コンポーネントによって停止または休されたときは停止しない場合がある。物理時間の進みレートは基礎となるハードウェアのそれと一致しているので、当事者は物理時間のレートを独自の処理時間またはメディア時間の進みレートと比較して処理レートを整合することができる。現メディア時間の照会を行い、ストリーム処理枯渇またはあふれに関するフィードバックを受けると、処理コンポーネントは処理レート整合のための調整ができるだけでなく、マスタまたは参照ストリームの所望ストリーム位置より進んでいるか、遅れているか（つまり、同期しているかどうか）を判断することもできる。
相関メディア時間	このプロパティはクロック・メカニズムの現メディア時間値および対応する参照時間値の両方を単一のアトミック・オペレーションで戻す。実施の形態では、ＰＣ時間が多数の異なる操作エンティティが共通に利用できる参照時間となっている。相関メディア時間を使用すると、異なるクロックを使用する処理コンポーネントは共通ＰＣ時間を使用して変換を行うことができるが、発生する誤差量は非常に少なく、多くの場合は、無視し得るほどである。正逆の変換は同じアトミック・オペレーションを使用するので、微小誤差は累積的でない。

【０２２６】　　【表１２】

（表４のつづき）

相関物理時間	このプロパティはクロック・メカニズムの現物理時間値および対応する参照時間値の両方を単一のアトミック・オペレーションで戻す。実施の形態では、ＰＣ時間が多数の異なる操作エンティティが共通に利用できる参照時間となっている。相関物理時間を使用すると、異なるクロックを使用する処理コンポーネントは共通ＰＣ時間を使用して変換を行うことができるが、発生する誤差量は非常に少なく、多くの場合は、無視し得るほどである。正逆の変換は同じアトミック・オペレーションを使用するので、微小誤差は累積的でない。
粒度と誤差	このプロパティはクロック・インクリメント粒度または解像度を戻し、クロックの各「刻時」がどれだけ正確または精細であるかを示す。実施の形態では、粒度値は各クロック刻時ごとに１００ナノ秒増分数になっている。クロックが１００ナノ秒解像度または粒度でまたはそれ以下で走行する場合は、値は１が戻されるだけである。粒度プロパティを使用すると、クロックのクライアントは解像度に応じて異なった反応をすることができる。このプロパティは誤差表示を１００ナノ秒増分値で行い、０は誤差量が最小であることを示している。

【０２２７】

【表１３】

（表４のつづき）

親	このプロパティはクロック・メカニズムを作成した処理コンポーネントのユニークな識別子を戻す。これにより、クライアントまたは処理コンポーネントはクロック・メカニズムが独自のクロック・メカニズムであるか、他の処理コンポーネントのクロック・メカニズムであるかを判断し、事情に応じて異なった反応をすることができる。実施の形態では、このプロパティはクロック・メカニズムが作成される基となった入力接続ピン・インスタンスまたはフィルタのファイル・オブジェクトへのハンドルである。
コンポーネント・状態	このプロパティは処理コンポーネント（つまり、フィルタ）の状態を反映し、クロック・メカニズムのクライアントは基礎となる処理コンポーネントが停止状態、実行状態、またはデータ取得状態にあるかを判断できるようにする。この場合も、このプロパティは基礎となるデータ・ストリームになにが起こったかに左右されるので、メディア時間と同じ働きをする。
関数テーブル	このプロパティはプロパティ集合インタフェースのサブセットを表す関数テーブルにクライアントがアクセスできるようにするデータ構造またはテーブルを示す。必然的に、このプロパティはプラットフォーム専用になっているが、これはフィルタ境界にまたがるカーネル・モード関数コールの扱い方がオペレーティング・システムが異なるごとに異なるためである。実施の形態では、ＮＴオペレーティング・システムを使用しているので、関数テーブルを使用すると、他のカーネル・モード・エンティティは高速化されたインタフェースをＤＰＣレベルで利用できる。これはＮＴカーネルＩＲＰを通してプロパティを使用する通常の方法とは対照的である。

【０２２８】この分野の当業者ならば理解されるように、表４にリストされている以外のプロパティまたは表４にリストされているプロパティのサブセットを使用して、ここで説明している相互接続フィルタ・システムに準拠する機能をもつクロック・メカニズム・プロパティ集合を作ることも可能である。さらに、プロパティ集合メカニズムには柔軟性があるので、独自のタイミング機能をもつフィルタ開発者は独自のＧＵＩＤをもつ別のプロパティ集合を追加することによって「必須」のプロパティ集合を拡張すると、より拡張した機能を得ることができる。集合メカニズムには、フィルタ開発者が最小限の機能量を実施して、フィルタをシステム・アーキテクチャに準拠させることができるという柔軟性があると同時に、フィルタ開発者がカストマイズして追加の機能を追加できるという無限性がある。

【０２２９】プロパティ集合によると、カーネル・モード・フィルタなどの他の処理コンポーネントは関係する時間プロパティについて照会することができるが、クロック・メカニズムはイベント集合もサポートするので、通知をイベント通知ＩＲＰを通して関係のクロック・メカニズム・クライアントに直接に送信することができる。ファイル・オブジェクトのハンドルはイベントを割込み可能にする一部としてクロック・メカニズムに「登録」しておくと、クロック・メカニズムはイベント通知をどこに送信すべきかを知ることができる。オプションとして、直接関数コール・ハンドルはＤＰＣ処理がカーネル・モード・エンティティ間で行えるように使用すると、パフォーマンスを向上することができる。

【０２３０】以下に示す表５は、コンプライアント・ドライバにサポートさせることができるイベント集合を構成する、起こり得る通知イベントのいくつかを示したものである。サード・パーティ制御エージェントはイベントを受け取るエンティティを、特定のフィルタ・グラフ・トポロジに基づいて相互接続または登録することができる。

【０２３１】

【表１４】

（表５）

	クロック・メカニズムのイベント
イベント	説明
インターバル・イベント	循環的に通知されるイベント。実施の形態では、特定のインターバルと特定のクロック時間は制御エージェントによって設定される。特定クロック時間で開始されると、循環的イベント通知は各インターバルが現われるごとに送信される。例えば、処理コンポーネントがビデオ・ストリームからのビデオ・フレームを毎秒３０フレーム単位でレンダリングする場合は、開始時間と１／３０秒のインターバルを指定し、イベント通知が同じ周期で送信されるようにすれば、ビデオ・フレームが正しくレンダリングされる。
位置イベント	特定のクロック時間に通知されるイベント。例えば、メディア・イベント・ストリームをレンダリングする処理コンポーネントまたはフィルタは、状況によっては、特定のクロック時間に音を演奏する必要が起こる場合がある。処理コンポーネントまたはフィルタは位置イベント通知を指定クロック時間に行うことを指定しておくと、音のレンダリングの開始を知ることができる。位置通知イベントを使用すると、停止または休止クロックのイベント通知が現在時刻に通知されるように指定することにより、クロックが実行状態になった後でいつ進行を開始するかを正確に知ることができる。

【０２３２】当業者ならば理解されるように、他のタイミング関係イベントをクロック・メカニズムに対して作成することが可能である。例えば、イベントはクロック・メカニズムがサポートできる異種の時間別（つまり、メディア時間または物理時間）に分類することが可能である。さらに、インターバル・イベントと位置イベント、さらには状態変化に基づく他のイベントのエレメントを組み合わせると、「ハイブリッド」イベントにすることも可能である。

【０２３３】次に、図２１（Ａ）を参照して説明すると、同図は、１つまたは２つ以上の「スレーブ」処理コンポーネントまたはフィルタを、「マスタ」メディア処理時間に基づいて同期化する方法を示している。ステップ３８０からスタートして、スレーブ処理コンポーネントまたはフィルタはステップ３８２でマスタ・クロック・メディア時間値を受信する。この受信は、スレーブ・フィルタまたはコンポーネントがマスタ・クロック・メカニズムに照会してメディア時間値を得ることによって行われるが、マスタ・クロック・メカニズムが通知をスレーブ処理コンポーネントまたはフィルタに送信することも可能である。図２１（Ａ）のフローチャートに示されている方法を図１９と図２０に示すフィルタ・グラフ・トポロジに適用すると、マスタ・クロック・メカニズムは３４４は、ビデオ・レンダラ・フィルタ３３２の入力ピン・インスタンス３３６とスレーブ処理コンポーネントして通信するファイル・オブジェクトによって表される。

【０２３４】なお、使用されるメディア時間値はマスタ・データ・ストリームの処理によって左右される。メディア・レンダリングが休止または停止されると、時間は同期をとる目的のための停止されることになる。

【０２３５】スレーブ処理コンポーネントまたはフィルタはステップ３８４で、マスタ・クロック・メディア時間値をスレーブ・データ・ストリーム・メディア時間値と比較して、スレーブ・メディア処理がどれだけ進んでいたか、あるいは遅れていたかを判断する。なお、メディア時間は実際のデータ・サンプルのストリーム内の時間インターバル情報に基づいているので、位置時間として見ることができる。言い換えれば、同期化とは、ストリーム内の同じ相対時間位置で２つのデータ・ストリームを処理することである。

【０２３６】ステップ３８８で終了する前に、スレーブ処理コンポーネントまたはフィルタはステップ３８６でマスタ・メディア処理時間に一致するようにスレーブ・メディア処理を調整する。この調整としては、スレーブ・メディア・ストリーム処理を減速または加速すること、基礎となるハードウェア・プロセッサのレートを変更すること、などがある。

【０２３７】メディア・ストリーム処理レートを減速することは、処理すべきサンプルを複製し、間欠的時間遅延を引き起こし、あるいは再サンプリングすることによって達成される。メディア・ストリーム処理を加速することは、処理すべきメディア・サンプルを省くか、再サンプリングすることによって達成される。この分野の当業者ならば理解されるように、変更が必要であると判断された後でメディア処理レートを調整する方法は多数存在する。

【０２３８】以上から理解されるように、図１９と図２０に示すトポロジ例では、入力ピン・インスタンス３３６はビデオ・レンダラ・フィルタ３３２のタイミング通知を受信する。ビデオ・レンダラ・フィルタ３３２は必要とされる調整を処理に行って図１９のビデオ・ストリームの処理を図２０のオーディオ・ストリームの処理と同期させる。

【０２３９】次に、図２１（Ｂ）のフローチャートを参照して、ライブまたはリアルタイム・オーディオ・ストリーム発生レートを処理コンポーネントまたはフィルタの処理レートとレート整合する処理ステップについて説明する。ステップ３９０でスタートした後、物理時間サンプルはステップ３９２で処理コンポーネントまたはフィルタで受信される。この場合も、これは、クロック・メカニズムに照会するか、あるいはタイミング・イベント通知を受信することによって行われる。図２０に示すように、最終的にはオーディオ・レンダラ・フィルタ３６４でレンダリングされるライブ・オーディオ・ストリーム３５２の処理では、関係する処理コンポーネントはイベント通知と制御情報が入力ピン・インスタンス３６８で受信されたときオーディオ・レンダラ・フィルタ３６８となる。また、クロック・メカニズム３４４は異なるタイミング情報を提供する。

【０２４０】ステップ３９２で受信された物理時間サンプルから、これらの時間サンプル間で処理されたデータ量をステップ３９４で使用すると、物理時間レートを計算または判断することができる。物理時間レートとは、オーディオ・ハードウェアが実際にデータをレンダリングするときのレートであり、データ・レンダリング・スループットを表し、フィルタ処理レートとも呼ばれている。ライブ・データ・ストリームのソースがフィルタ処理レートよりも早い発生レートでデータ・サンプルを発生すると、余剰データが発生する。発生レートがフィルタ処理レート以下であれば、データ処理のギャップまたは枯渇が発生する。どちらの場合も、パフォーマンスが低下することになる。

【０２４１】データ・サンプルを発生するプロセッサの実際の処理レート、つまり、発生レートは、メディアまたはデータ・サンプル自体から判断することができる。メディア・サンプルは規則または実際のタイムスタンプ情報の形態になった時間インターバル情報をもっているので、メディア時間サンプルはステップ３９６でメディア・ストリームから計算することができる。これにより、メディア・サンプルを作成したハードウェアの処理レートを表す発生レートは、一定のデータ処理量を使用し、この処理量をメディア時間サンプルから判断されたそのデータ処理時間で除すことにより計算することができる。

【０２４２】ステップ４００で終了する前に、メディア・ストリーム処理はステップ３９８で発生レートをフィルタ処理レートに一致させるように調整される。この場合も、調整はレンダリングするサンプルを省き、サンプル・レンダリング間に時間遅延を追加し、この分野で公知の他の方法によって行うことができる。

【０２４３】次に、図２１（Ｃ）に示すフローチャートを参照して、クロック・メカニズムを使用して変換を行うための処理ステップについて説明する。ステップ４０２でスタートした後、処理コンポーネントまたはフィルタはステップ４０４でクロック・メカニズムから相関時間値を受信する。なお、処理コンポーネントまたはフィルタはＰＣクロックのように、相関時間値の参照時間値部分を生成する参照時間クロックにアクセスすることができる。メディア時間データは、ＰＣクロックまたは以前に受信された相関時間値に入っている他の参照時間値からメディア時間値を減算することによってステップ４０６で計算される。メディア時間データは変換のために処理コンポーネントまたはフィルタによって使用され、ＰＣクロック時間からのメディア時間の変化を示している。

【０２４４】変換を行うためには、処理コンポーネントまたはフィルタはステップ４０８で現在のＰＣクロック時間値（または他の参照時間値）を取得する。処理コンポーネントまたはフィルタでの現在メディア時間は、ステップ４１０で現在ＰＣクロック時間値とマスタ・メディア・デルタを使用して変換することができる。メディア時間デルタは、参照時間を基準にしたＰＣクロック時間からのメディア時間の変化を示しているので、これを現在ＰＣクロック時間値に加えるだけで、タイムスタンプ情報の正しい「変換」または新時間値を得ることができる。ステップ４１０で変換プロセスを終了すると、処理はステップ４１２で終了する。この通信がＤＳＰなどのリモート・プロセッサ上の２つのコンポーネント間で行われる場合は、参照時間値はＰＣ時間ではなく、ローカル・ホスト時間となる。

【０２４５】図２１（Ｃ）のフローチャートに示す変換のメソッドを相互接続カーネル・モード・フィルタに適用して図１９と図２０に示すライブ・オーディオとビデオ・ストリームを処理し、レンダリングする場合は、ステップ４０４でクロック・メカニズム３４４から相関時間値を受信するのはビデオ・レンダラ・フィルタ３３２上の入力ピン・インスタンス３３６である。最後に、ラ

イブ・ビデオ・ストリーム３２０上のタイムスタンプの計算と変換を行うか、あるいはクロック・メカニズム３４４のメディア時間に基づいて変換参照時間を作成するのは、ビデオ・レンダラ・フィルタ３３２である。

【０２４６】この分野の当業者ならば理解されるように、本明細書に開示されている相互接続フィルタのシステムにおいては、均一バッファ割当て機能とタイミング機能は実施上の問題解決の要件に応じて、処理効率向上のために組み合わせて使用することも、単独で使用することも可能である。さらに、タイミング機能は、相互接続フィルタのシステムから独立して使用することも可能である。

【０２４７】この分野の精通者ならば理解されるように、本発明の種々方法はコンピュータ・プログラム・コード手段として、磁気ディスク、ＣＤ－ＲＯＭ、およびこの分野で共通している他の媒体やまだ未開発の他の共通媒体などの、コンピュータ読取り可能媒体上にストアされるコンピュータ命令として組み込んでおくことが可能である。さらに、コンピュータ・ハードウェア・メモリに置かれる重要なデータ構造は、上記のようなコンピュータ・プログラム・コード手段をオペレーションにより作成することができる。

【０２４８】本発明は本発明の基本的特徴の精神から逸脱しない限り、他の実施形態で実現することも可能である。上述してきた各種実施の形態はすべての点で説明した通りであるが、これらの実施の形態に限定されるものではない。従って、本発明の範囲は上述してきた説明によってではなく、特許請求の範囲に記載されている事項によってのみ限定されるものである。特許請求の範囲の等価技術の意味と範囲に属する一切の変更は本発明の範囲に属するものである。

【図面の簡単な説明】

【図１】制御エージェントの指示を受けてサウンド・データをディスク・ファイルから持ち込み、サウンド・データをなんらかの形態で処理し、サウンド・データをレンダリングしてスピーカから再生させる相互接続フィルタとドライバのシステムを示す従来技術のデータ・フロー図である。

【図２】図１に示すシステムと目的が同じであり、サウンド・データをディスク・ドライバから読み取り、そのデータを処理し、そのデータをレンダリングしてスピーカから聞こえるようにし、処理フィルタとレンダリングは、この場合も制御エージェントの指示を受けて相互接続カーネル・モード・ドライバによって処理されるようした本発明によるシステムを示す図である。

【図３】オペレーティング・システムで作成され、使用されるドライバ・オブジェクト、デバイス・オブジェクトおよびファイル・オブジェクト間の関係を示す垂直関係モデルを示す図である。

【図４】ドライバ・オブジェクトのロジック・ブロック図であり、本発明のシステムに従ってメッセージを適切なプロセス・ハンドリング・コードへ転送し、新しいファイル・オブジェクトの作成をバリデーションするためのデータ構造およびプログラム・コードとの論理的関係を示す図である。

【図５】デバイス・オブジェクトのロジック・ブロック図であり、本発明のシステムに従ってメッセージを適切なプロセス・ハンドリング・コードへ転送し、新しいファイル・オブジェクトの作成をバリデーションするためのデータ構造およびプログラム・コードとの論理的関係を示す図である。

【図６】ファイル・オブジェクトのロジック・ブロック図であり、本発明のシステムに従ってメッセージを適切なプロセス・ハンドリング・コードへ転送し、新しいファイル・オブジェクトの作成をバリデーションするためのデータ構造およびプログラム・コードとの論理的関係を示す図である。

【図７】ルーチングとバリデーション・コンポーネントリの初期セットアップとカーネル・モード・ドライバによる入出力メッセージの処理を示すフローチャートである。

【図８】制御エージェントの処理、ルーチングとバリデーション・メカニズム、および新しいファイル・オブジェクトを作成する具体的な作成ハンドラ・ルーチンを示す詳細フローチャートである。

【図９】オペレーティング・システムでファイル・オブジェクト構造を利用して、接続を標準化された方法で行う接続フィルタ間の水平関係を示すロジック図である。

【図１０】図９のカーネル・モード・フィルタまたはドライバを作成し、接続するためにユーザ・モードの制御エージェントによってとられる処理ステップを示すフローチャートであり、制御エージェントから受け取った入出力要求を処理するために接続を行い、その処理が異なるドライバ（フィルタ）間で続けられる様子を示している。

【図１１】ユーザ・モードの制御エージェントの指示を受けてカーネル・モード・フィルタのチェインを作成するために使用され、サウンド・データをハード・ドライブから読み取り、カーネル・モード・フィルタでそのデータを処理し、そのデータをレンダリングしてスピーカから聞こえるようにするシステムを実装するためのカーネル・モード・ドライバと接続を示す概要ロジック図である。

【図１２】ユーザ・モードの制御エージェントの指示を受けてカーネル・モード・フィルタのチェインを作成するために使用され、サウンド・データをハード・ドライブから読み取り、カーネル・モード・フィルタでそのデータを処理し、そのデータをレンダリングしてスピーカから聞こえるようにするシステムを実装するためのカーネル・モード・ドライバと接続を示す概要ロジック図で

ある。

【図１３】図１１と図１２に示すシステム用に相互接続カーネル・モード・ドライバを作成するための処理ステップを示すフローチャートである。

【図１４】バッファ割当てメカニズムがどのような働きをするかを示す図であり、割当てられたバッファ・フレームがある処理コンポーネントから別の処理コンポーネントへ渡されるときの、バッファ・フレームの論理的構成と処理を示す。

【図１５】バッファ割当てメカニズムがどのような働きをするかを示す図であり、相互接続カーネル・モード・フィルタのシステムにおいて入力ピン・インスタンスを表すファイル・オブジェクトの「子」であるファイル・オブジェクトとして表されるバッファ・アロケータを示している。図１４と図１５はどちらも同じフィルタ・グラフ・トポロジを示している。

【図１６】図１１および図１２に示すシステムの遷移において、バッファ・フレームの割当てを制御するバッファ・アロケータを利用して行われるバッファ割当てを示す図である。

【図１７】相互接続カーネル・モード・フィルタのチェインを通してデータをディスク・ドライバから持ち込んで、サウンド処理ハードウェア上でデータをレンダリングする処理ステップを示すフローチャートであり、具体的には、バッファ・アロケータのオペレーションと図１６に示すシステムでバッファ間で実際に行われるデータ転送を示している。

【図１８】２つのライブ・データ・ストリームが単一のマスタ・クロック・メカニズムとどのようにして同期化されるかを示すロジック・ブロック図である。

【図１９】図１６を参照して詳しく説明されている相互接続フィルタ・システムを使用して実施されている図１８のライブ・オーディオ・システムを示すロジック・ブロック図であり、図１９はマスタ・クロック同期化信号を受信するライブ・ビデオ・ストリーム・レンダリング・フィルタを示す。

【図２０】図１６を参照して詳しく説明されている相互接続フィルタ・システムを使用して実施されている図１８のライブ・オーディオ・システムを示すロジック・ブロック図であり、図２０は両方のストリームを一緒に同期化し、ライブ・オーディオ・データを実際のオーディオ・レンダリング・ハードウェアとレート整合するマスタ・クロック・メカニズムをもつライブ・オーディオ・レンダリング・システムを示す図である。

【図２１】（Ａ）ないし（Ｃ）は、複数のデータ・ストリームのデータ処理を同期化し、あるデータ・ストリームをハードウェア・レンダラの物理的機能とレート整合し、あるタイムベースを共通タイムベースを使用して別のタイムベースに変換するためにクロック・メカニズムがどうように使用されるかを示す図であり、（Ａ）は同期化処理ステップを示すフローチャート、（Ｂ）はレート整合処理ステップを示すフローチャート、（Ｃ）は変換処理ステップを示すフローチャートである。

【符号の説明】

４４，１７０，３４２　制御エージェント
４６，２６２　ディスク・ドライブ
４８，１８２　ディスク・ドライバ
５０　リーダ・ドライバ
５２　デコンブレッサ・ドライバ
５４　効果フィルタ
５６　効果プロセッサ
５８　レンダリング・ドライバ
６２，３０２　スピーカ
６４，７６，８０　ドライバ・オブジェクト
６６　デバイス・オブジェクト
７０，７２，７４，９０　ファイル・オブジェクト
７８　汎用マルチプレクシング・ディスパッチ機能
８２　デバイス・エクステンション・エリア
８４，１００　ファイル・タイプ・バリデーション・テーブル
８６，１０２　ファイル・オブジェクト・タイプ
８８　作成ハンドラ
９２　ファイル・コンテキスト・エリア
９４　ＩＲＰ要求ハンドラ・テーブル
９６　ＩＲＰ要求
９８　ハンドラ
１０４　参照
１４６　フィルタＡ
１４８　フィルタＢ
１５４，１８８，２０２，２０６，２４２，２４８，３２８，３３６，３６２，３６８　入力ピン・インスタンス
１５８，１９２，２０４，２０８，２４０，２５０，３２６，３３４，３６０，３６６　出力ピン・インスタンス
１７２　リーダ・フィルタ
１７４　デコンブレッサ・フィルタ
１７６　効果フィルタ
１７８　サウンド・レンダリング・フィルタ
２１０　シンク処理コンポーネント
２１２　バッファ・アロケータ・メカニズム
２１４，２３８　ソース処理コンポーネント
２１６，２２２　バッファ・アロケータ参照
２２０　変換処理コンポーネント
２２４ａ　バッファ・フレーム
２４４　変換フィルタ
２４６　シンク・フィルタ
２５２，２６４，２６８　バッファ・アロケータ
２５６　システム・メモリ
２５８　サウンド・カード・メモリ

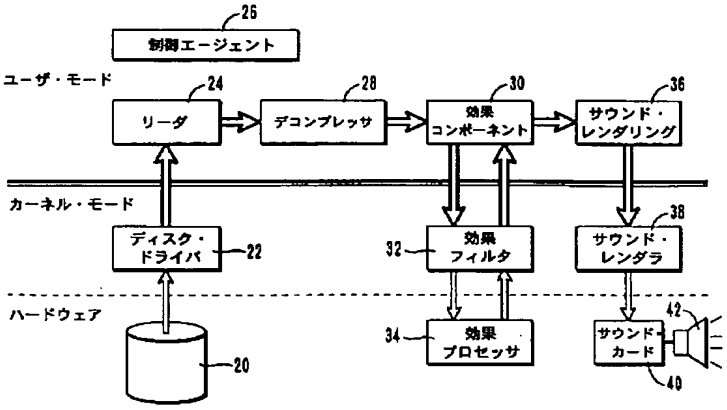
89

２６０　システム・メモリ
２９６　オーディオ・レンダラ
２９８，３５２　ライブ・オーディオ・ストリーム
３０６　ビデオ・レンダラ
３０８，３２０　ライブ・ビデオ・ストリーム
３１２，３５０　モニタ
３１６　マスタ・クロック・メカニズム
３２２　ビデオ・リーダ・フィルタ

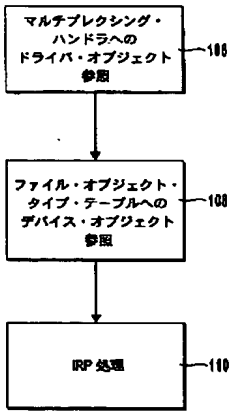
90

＊３２８　入力ピン・インスタンス
３３０　ビデオ・デコンプレッサ・フィルタ
３３２　ビデオ・レンダリング・フィルタ
３４４　クロック・メカニズム
３４６　ビデオ・ハードウェア
３５４　オーディオ・リーダ・フィルタ
３５８　オーディオ・デコンプレッサ・フィルタ
＊　３６４　オーディオ・レンダラ・フィルタ

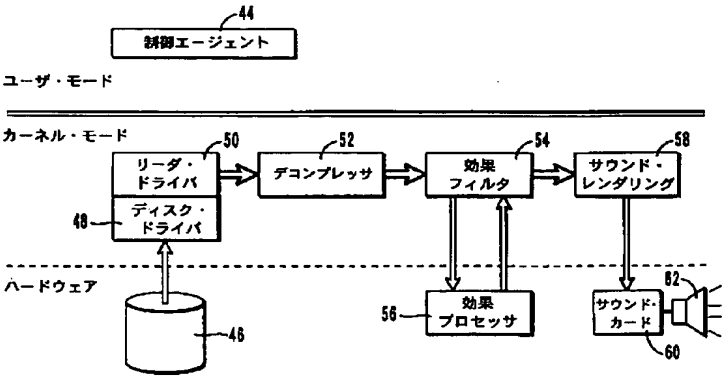
【図１】



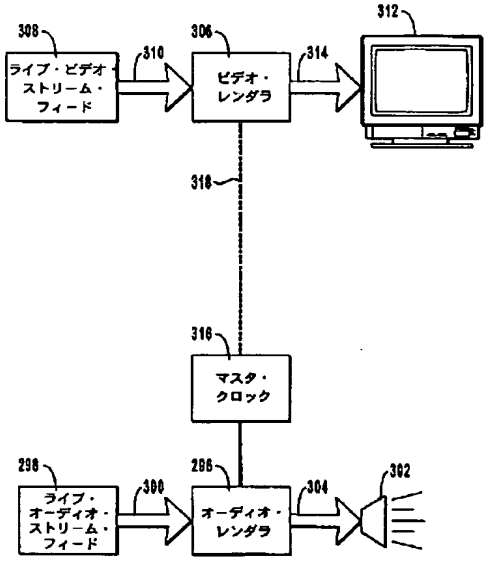
【図７】



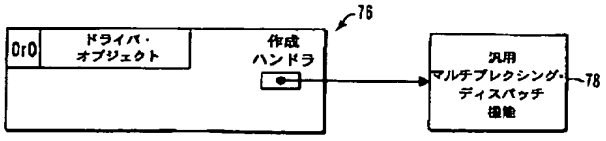
【図２】



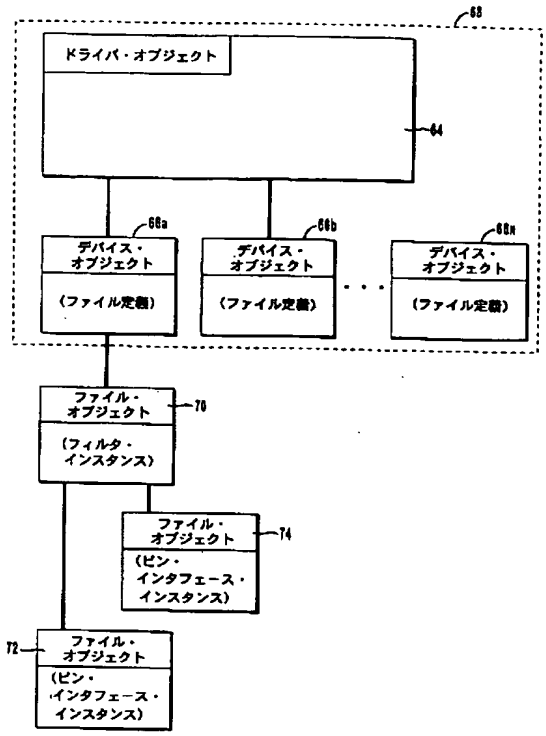
【図１８】



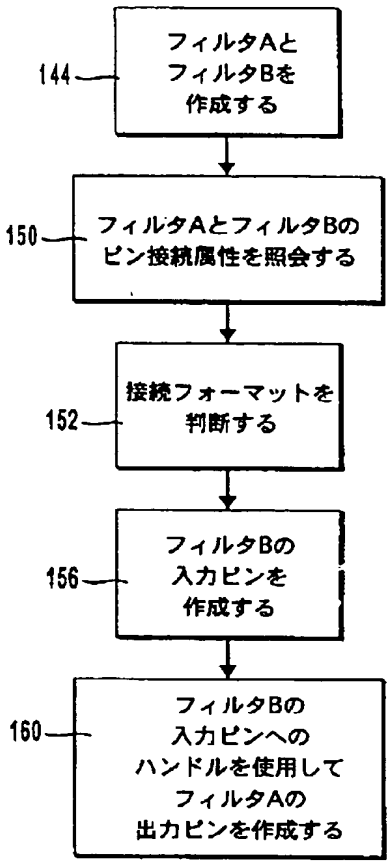
【図４】



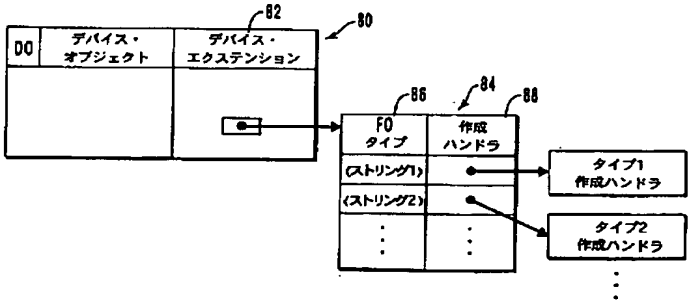
【図３】


68
ドライバ・オブジェクト
64
66a
66b
66n
デバイス・オブジェクト
(ファイル定義)
デバイス・オブジェクト
(ファイル定義)
デバイス・オブジェクト
(ファイル定義)
ファイル・オブジェクト
70
(フィルタ・インスタンス)
ファイル・オブジェクト
74
(ピン・インタフェース・インスタンス)
72
ファイル・オブジェクト
(ピン・インタフェース・インスタンス)

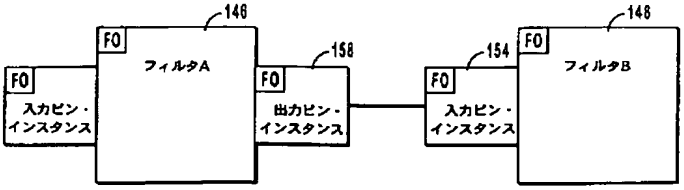
【図１０】


144
フィルタAとフィルタBを作成する
150
フィルタAとフィルタBのピン接続属性を照会する
152
接続フォーマットを判断する
156
フィルタBの入力ピンを作成する
160
フィルタBの入力ピンへのハンドルを使用してフィルタAの出力ピンを作成する

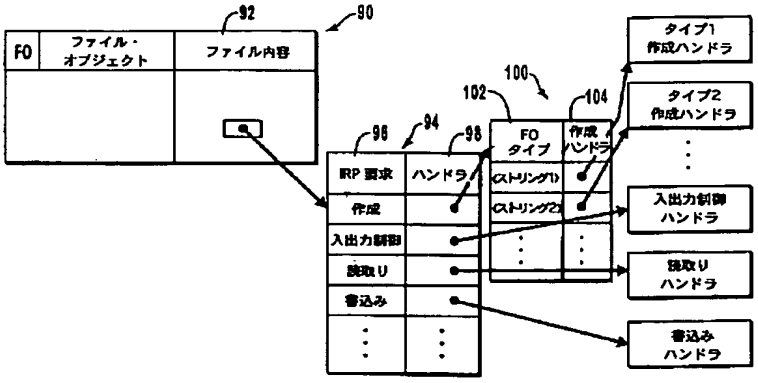
【図５】


82
80
DO
デバイス・オブジェクト
デバイス・エクステンション
86
84
88
FO タイプ
作成ハンドラ
(ストリング1)
(ストリング2)
タイプ1 作成ハンドラ
タイプ2 作成ハンドラ

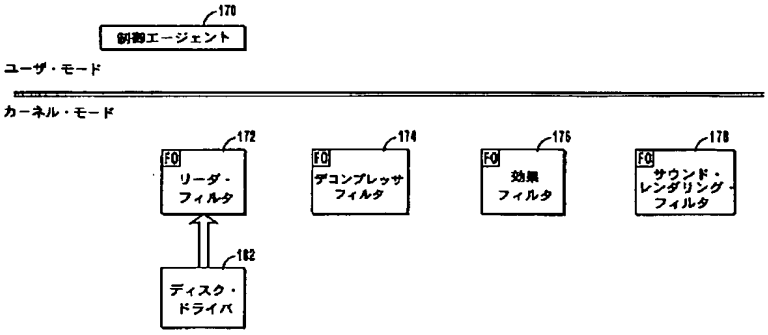
【図９】


146
FO
フィルタA
FO
入力ピン・インスタンス
158
FO
出力ピン・インスタンス
154
FO
入力ピン・インスタンス
148
FO
フィルタB

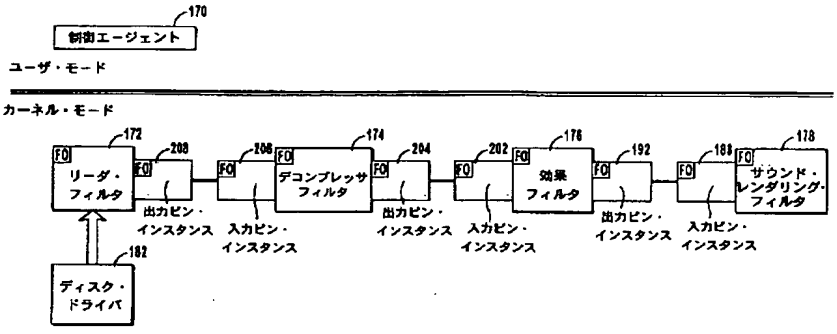
【図６】


92
90
FO
ファイル・オブジェクト
ファイル内容
100
102
104
96
94
98
FO タイプ
作成ハンドラ
IRP 要求
ハンドラ
<ストリング1>
<ストリング2>
作成
入出力制御
読取り
書込み
タイプ1 作成ハンドラ
タイプ2 作成ハンドラ
入出力制御ハンドラ
読取りハンドラ
書込みハンドラ

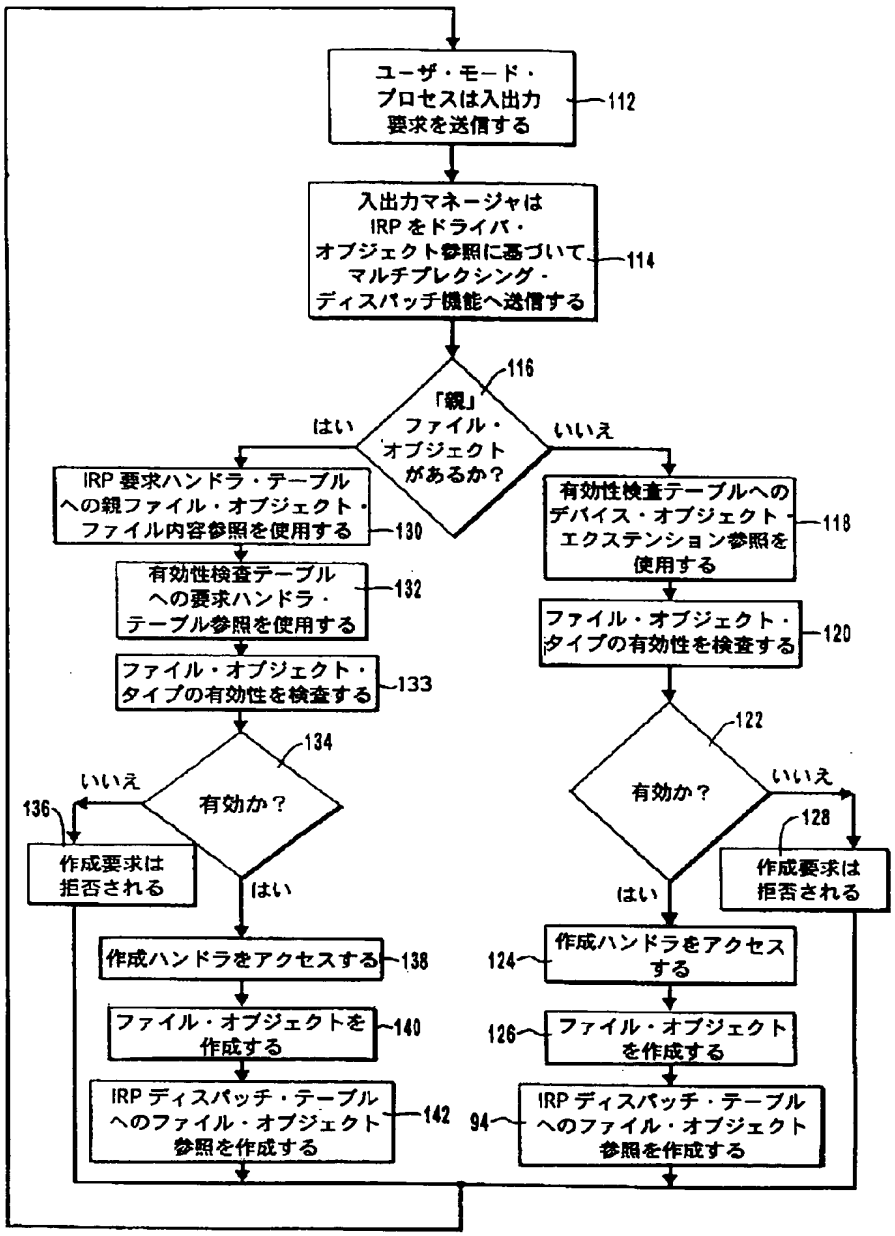
【図１１】


170
制御エージェント
ユーザ・モード
カーネル・モード
172
FO
リーダ・フィルタ
174
FO
デコンプレッサ フィルタ
176
FO
効果 フィルタ
178
FO
サウンド・レンダリング・フィルタ
182
ディスク・ドライバ

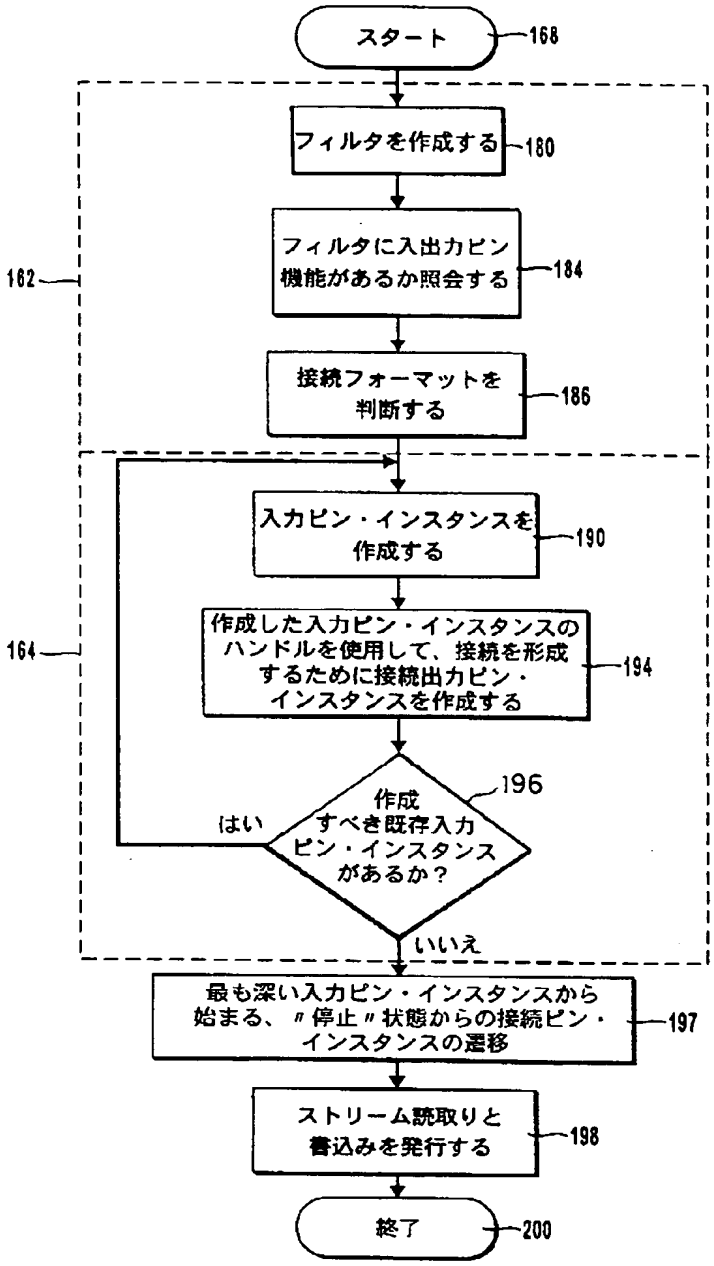
【図１２】


170
制御エージェント
ユーザ・モード
カーネル・モード
172
FO
リーダ・フィルタ
208
FO
出力ピン・インスタンス
206
FO
入力ピン・インスタンス
174
FO
デコンプレッサ フィルタ
204
FO
出力ピン・インスタンス
202
FO
入力ピン・インスタンス
176
FO
効果 フィルタ
192
FO
出力ピン・インスタンス
188
FO
入力ピン・インスタンス
178
FO
サウンド・レンダリング・フィルタ
182
ディスク・ドライバ

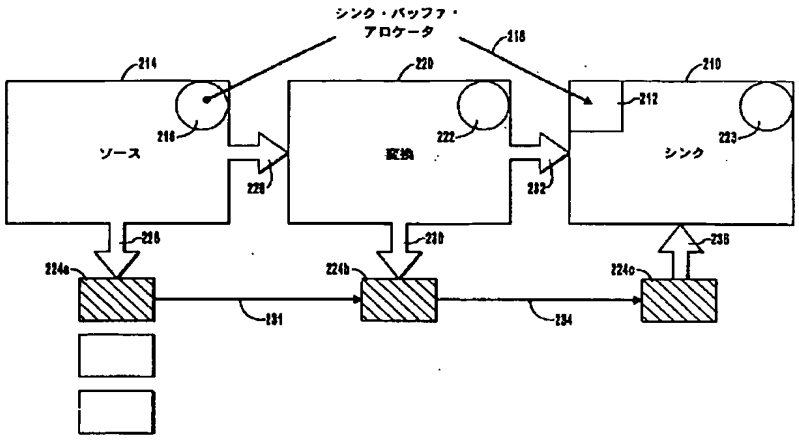
【図８】


ユーザ・モード・プロセスは入出力要求を送信する
112
入出力マネージャはIRPをドライバ・オブジェクト参照に基づいてマルチプレクシング・ディスパッチ機能へ送信する
114
「親」ファイル・オブジェクトがあるか？
116
はい
いいえ
IRP要求ハンドラ・テーブルへの親ファイル・オブジェクト・ファイル内容参照を使用する
130
有効性検査テーブルへのデバイス・オブジェクト・エクステンション参照を使用する
118
有効性検査テーブルへの要求ハンドラ・テーブル参照を使用する
132
ファイル・オブジェクト・タイプの有効性を検査する
120
ファイル・オブジェクト・タイプの有効性を検査する
133
有効か？
134
122
いいえ
はい
136
作成要求は拒否される
128
作成要求は拒否される
作成ハンドラをアクセスする
138
124
作成ハンドラをアクセスする
ファイル・オブジェクトを作成する
140
126
ファイル・オブジェクトを作成する
IRPディスパッチ・テーブルへのファイル・オブジェクト参照を作成する
142
94
IRPディスパッチ・テーブルへのファイル・オブジェクト参照を作成する

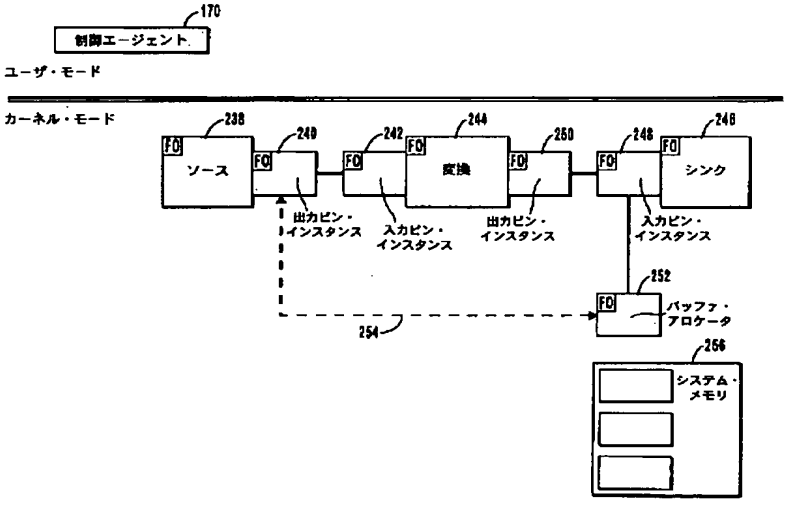
【図１３】


スタート
168
フィルタを作成する
180
フィルタに入出力ピン機能があるか照会する
184
162
接続フォーマットを判断する
186
入力ピン・インスタンスを作成する
190
作成した入力ピン・インスタンスのハンドルを使用して、接続を形成するために接続出力ピン・インスタンスを作成する
194
164
作成すべき既存入力ピン・インスタンスがあるか？
196
はい
いいえ
最も深い入力ピン・インスタンスから始まる、〃停止〃状態からの接続ピン・インスタンスの遷移
197
ストリーム読取りと書込みを発行する
198
終了
200

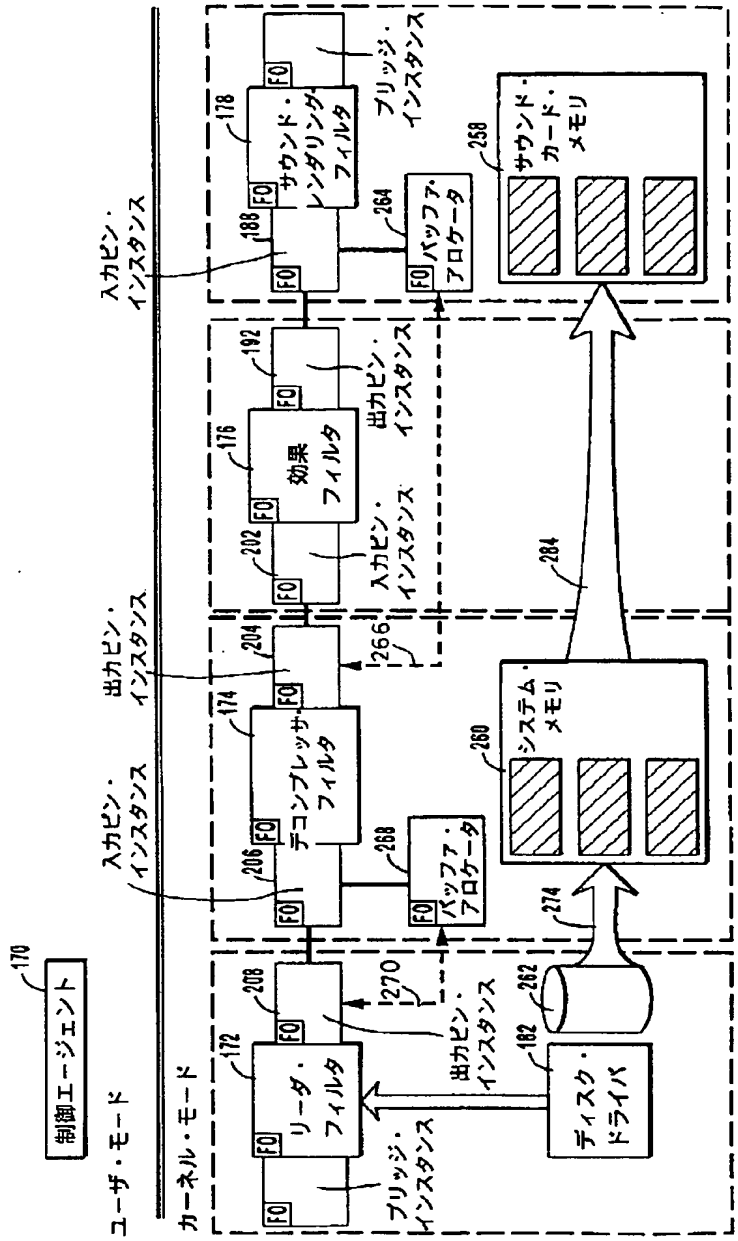
【図１４】


シンク・バッファ・
アロケータ
218
214
220
210
212
216
ソース
222
変換
223
シンク
228
232
226
230
236
224a
224b
224c
231
234

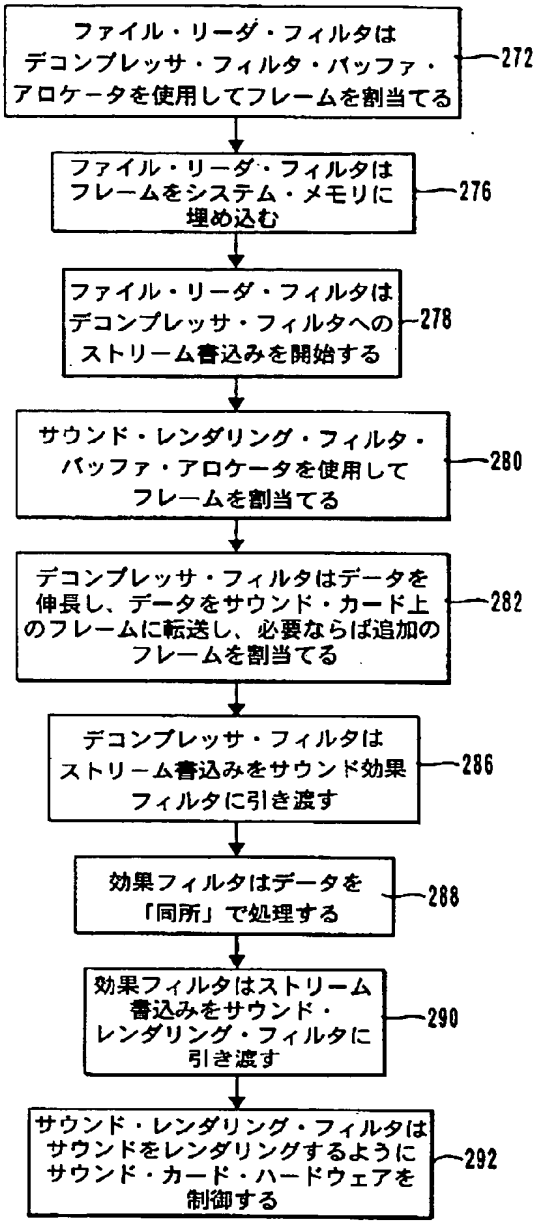
【図１５】


170
制御エージェント
ユーザ・モード
カーネル・モード
238
240
242
244
250
248
246
FO
ソース
変換
シンク
出力ピン・
インスタンス
入力ピン・
インスタンス
出力ピン・
インスタンス
入力ピン・
インスタンス
252
バッファ・
アロケータ
254
256
システム・
メモリ

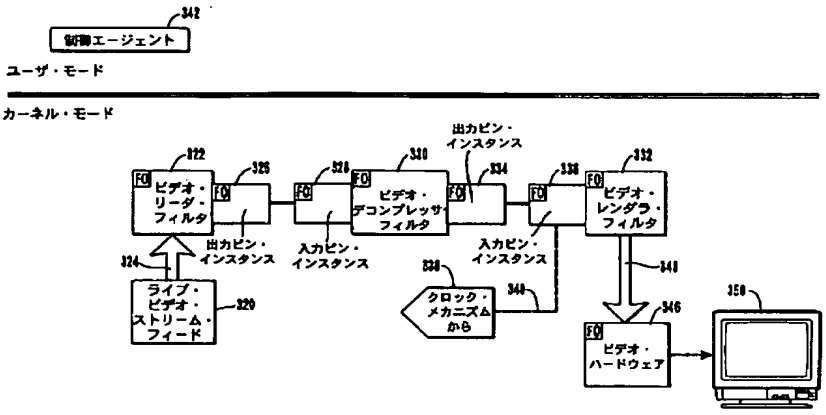
【図１６】


制御エージェント
170
ユーザ・モード
カーネル・モード
入力ピン・インスタンス
出力ピン・インスタンス
入力ピン・インスタンス
出力ピン・インスタンス
入力ピン・インスタンス
出力ピン・インスタンス
ブリッジ・インスタンス
リーダ・フィルタ
172
デコンプレッサ・フィルタ
174
効果フィルタ
176
サウンド・レンダリング・フィルタ
178
FO
206
204
202
192
188
208
270
182
262
268
266
264
258
260
274
284
バッファ・アロケータ
システム・メモリ
サウンド・カード・メモリ
ディスク・ドライバ

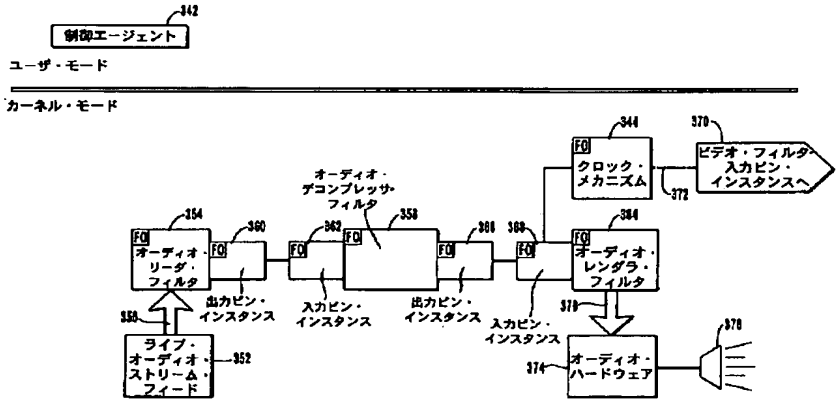
【図１７】


ファイル・リーダ・フィルタはデコンプレッサ・フィルタ・バッファ・アロケータを使用してフレームを割当てる
272
ファイル・リーダ・フィルタはフレームをシステム・メモリに埋め込む
276
ファイル・リーダ・フィルタはデコンプレッサ・フィルタへのストリーム書込みを開始する
278
サウンド・レンダリング・フィルタ・バッファ・アロケータを使用してフレームを割当てる
280
デコンプレッサ・フィルタはデータを伸長し、データをサウンド・カード上のフレームに転送し、必要ならば追加のフレームを割当てる
282
デコンプレッサ・フィルタはストリーム書込みをサウンド効果フィルタに引き渡す
286
効果フィルタはデータを「同所」で処理する
288
効果フィルタはストリーム書込みをサウンド・レンダリング・フィルタに引き渡す
290
サウンド・レンダリング・フィルタはサウンドをレンダリングするようにサウンド・カード・ハードウェアを制御する
292

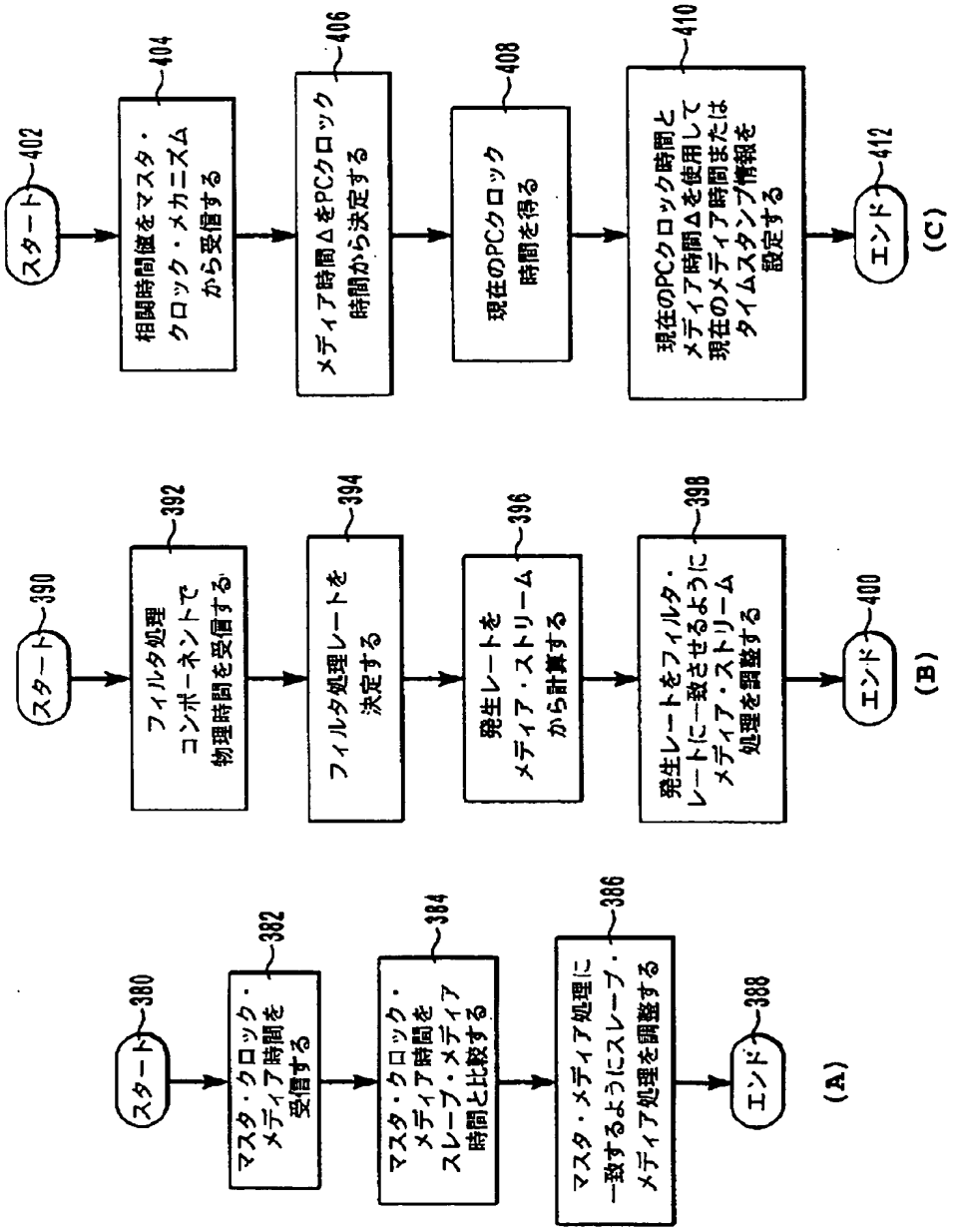
【図１９】


制御エージェント
342
ユーザ・モード
カーネル・モード
ビデオ・リーダ・フィルタ
322
325
ビデオ・デコンプレッサ・フィルタ
328
330
出力ピン・インスタンス
334
ビデオ・レンダラ・フィルタ
338
332
入力ピン・インスタンス
入力ピン・インスタンス
出力ピン・インスタンス
324
ライブ・ビデオ・ストリーム・フィード
320
クロック・メカニズムから
336
340
348
346
350
ビデオ・ハードウェア

【図２０】


制御エージェント
342
ユーザ・モード
カーネル・モード
344
370
クロック・メカニズム
ビデオ・フィルタ入力ピン・インスタンスへ
372
オーディオ・デコンプレッサ・フィルタ
354
360
362
358
380
368
384
オーディオ・リーダ・フィルタ
オーディオ・レンダラ・フィルタ
出力ピン・インスタンス
入力ピン・インスタンス
出力ピン・インスタンス
入力ピン・インスタンス
350
376
378
ライブ・オーディオ・ストリーム・フィード
352
374
オーディオ・ハードウェア

【図２１】



フロントページの続き

(72)発明者 ブライアン エイ. ウッドラフ
アメリカ合衆国 98045 ワシントン州 ニュー ベンド サウス ウエスト 10テ ィーエイチ ストリート 1020

(72)発明者 トーマス ジェイ. エロウルケ
フィンランド 90800 オウル リコニナ ルヤティエ 29

【外国語明細書】

１．Title of the Invention

Method and Computer Program Product for Synchronizing the Processing of Multiple Data Streams and Matching Disparate Processing Rates Using a Standardized Clock Mechanism

２．Claims

1. A method for interconnecting software drivers to allow efficient kernel mode processing of data and having a standardized way of providing timing information for synchronizing different data streams or rate matching different hardware clocks, the method comprising the steps of:

opening one or more kernel mode drivers;

forming one or more connection pin instances for connecting the drivers, each connection pin instance hierarchically related to one of said one or more drivers and used for data transmission between said one or more drivers;

creating one or more clock mechanisms for rate matching and stream synchronization, each clock hierarchically related to one of said one or more connection pin instances and providing a data stream time and a physical time based on an underlying hardware oscillator; and

interconnecting said one or more connection pin instances so as to provide a continuous data flow path through the one or more drivers residing in kernel mode, said clocks making timing information available for rate matching between different clock oscillations and for synchronizing different data streams.

2. A method as recited in claim 1 wherein each connection pin instance is represented by a file object and the hierarchal relationship is created by specifying the related driver, the driver referenced as a file object of an I/O device available on the system, during connection pin instance file object creation and each clock mechanism is represented by a file object and the hierarchal relationship with a connection pin instance is created by specifying the related connection pin instance file object as a parent during the clock file object creation.

3. A method as recited in claim 1 further comprising the step of querying each of said connection pin instances on each of said one or more drivers to determine the availability of clock mechanisms in order to determine whether to instantiate a clock mechanism for synchronization or rate matching on a particular connection pin instance prior to interconnecting said one or more connection pin instances.

4. A method as recited in claim 1 wherein at least one of said one or more connection pin instances on said one or more drivers support at least one predefined set of properties, set of methods, and set of events to indicate to a third party component the availability of a clock mechanism on the at least one connection pin instance and for allowing a third party component to form said clock mechanism on respective connection pin instances and the clocks support at least one predefined set of properties, set of methods, and set of events to control said respective clocks as determined by the third party component.

5. A computer program product comprising:

a computer usable medium having computer readable program code means embodied in said medium for providing a standardized interconnection mechanism to other components in kernel mode, said computer readable program code means comprising;

program code means for forming connection pin instances, said connection pin instances used for transmitting data to or from other components in kernel mode and capable of interconnection with other connection pin instances located on other components; and

program code means for forming clock mechanisms on some of said connection pin instances, said clock mechanisms providing a positional time value that reflects a current position in a stream of data having time interval information associated therewith.

6. A computer program product as recited in claim 5 wherein the clock mechanism further provides a correlated time value that comprises a reference time value based on a common hardware oscillator available to multiple components and said positional time value in an atomic operation so as to allow other components to perform time translations using said common hardware oscillator as a reference.

7. A computer program product as recited in claim 5 wherein the clock mechanism further provides a correlated time value that comprises a reference time value based on a common hardware oscillator available to multiple components and designated time value in an atomic operation so as to allow other components to perform time translations using said common hardware oscillator as a reference.

8. A computer program product as recited in claim 5 wherein the clock mechanism further provides a physical time value that is based on a hardware oscillator.

9. A computer program product as recited in claim 8 wherein the clock mechanism further provides a correlated physical time value that comprises a reference time value based on a common hardware oscillator available to multiple components and said physical time value in an atomic operation so as to allow other components to perform time translations using said common hardware oscillator as a reference.

10. A computer program product as recited in claim 5 wherein said positional time value is accessed by other components as part of a property set implemented in said program code means.

11. A computer program product as recited in claim 5 wherein the clock mechanism further provides notifications of relevant events to other components.

12. A computer program product as recited in claim 5 wherein the clock mechanism further provides notifications based upon said positional time of relevant events to other components so as to provide useful synchronization points.

13. A method for synchronizing multiple streams of data comprising the steps of:

creating a master clock reference based on a reference data stream of data samples, each data sample having time interval information indicating position within the reference data stream, the master clock reference created from the time interval information;

processing one or more other data streams based on the master clock reference so that the other data streams are synchronized with the processing of the reference data stream.

14. A method as recited in claim 13 wherein the master clock reference is accessed as part of a property set.

15. A method as recited in claim 13 wherein the master clock reference is part of a selectively instantiable clock mechanism that is created in response to a request by a third party agent.

16. A method as recited in claim 13 wherein the time interval information is provided by timestamp information on the media samples.

17. A method as recited in claim 13 wherein the time interval information is provided media stream convention.

18. A method for rate matching a data stream processed through a processing component to a hardware processor comprising the steps of:

providing, to the processing component, a physical time reference based on the hardware oscillations of a hardware processor; and

adjusting the rate of processing at the processing component of a data stream to match the hardware processor processing rate based on the physical time reference, the data stream comprised of samples having time interval information associated therewith.

19. A method as recited in claim 18 wherein the physical time reference is accessed as part of a property set.

20. A method as recited in claim 18 wherein the physical time reference is part of a selectively instantiable clock mechanism that is created in response to a request by a third party agent.

21. A method as recited in claim 18 wherein the time interval information is provided by timestamp information on the data samples.

22. A method as recited in claim 18 wherein the time interval information is provided data stream convention.

23. A method for synchronizing multiple media streams comprising the steps of:

creating a master clock reference based on a reference media stream of media samples, each sample having timestamp information indicating position within the reference media stream, the master clock reference created from the timestamp information;

processing one or more other media streams based on the master clock reference so that the other media streams are synchronized with the processing of the reference media stream.

24. A method as recited in claim 23 wherein the master clock reference is accessed as part of a property set.

25. A method as recited in claim 23 wherein the master clock reference is part of a selectively instantiable clock mechanism that is created in response to a request by a third party agent.

26. A method for rate matching a media stream processed through a processing component to a hardware renderer comprising the steps of:

providing, to the processing component, a physical time reference based on the hardware oscillations of a hardware renderer; and

adjusting the rate of processing at the processing component of a media stream to match the hardware renderer processing rate based on the physical time reference, the media stream comprised of media samples having time interval information associated therewith.

27. A method as recited in claim 26 wherein the physical time reference is accessed as part of a property set.

28. A method as recited in claim 26 wherein the physical time reference is part of a selectively instantiable clock mechanism that is created in response to a request by a third party agent.

29. A method as recited in claim 26 wherein the time interval information is provided by timestamp information on the media samples.

30. A method as recited in claim 26 wherein the time interval information is provided media stream convention.

31. A method for translating a positional time value in a first component based on a positional time value in a second component and utilizing a common hardware oscillator comprising the steps of:

receiving, at a first component from a second component, a correlated time value in a single operation, the correlated time value comprising a positional time value based on the position within a data stream having time interval information associated therewith and a common time value based a common hardware oscillator;

querying, by the first component, the current value of the common hardware oscillator; and

translating, by the first component, the positional time value in the first component based on the current value of the common hardware oscillator and the correlated time value received from the second component.

32. A method for translating a designated time value in a first component based on a designated time value in a second component and utilizing a common hardware oscillator comprising the steps of:

receiving, at a first component from a second component, a correlated time value in a single operation, the correlated time value comprising a designated time value and a common time value based a common hardware oscillator;

querying, by the first component, the current value of the common hardware oscillator; and

translating, by the first component, the designated time value in the first component based on the current value of the common hardware oscillator and the correlated time value received from the second component.

33. A method as recited in claim 32 wherein the designated time value is a positional time value based on the position within a data stream having time interval information associated therewith.

34. A method as recited in claim 32 wherein the designated time value is a physical time value based upon a hardware oscillator.

35. A kernel mode data processing system allowing synchronous processing of two data streams comprising:

a first data source;

a first plurality of kernel mode data processing components including an originating component and a terminating component, the originating component reading data samples of a first data stream originating from the first data source, and at least one of the processing components having a master clock component associated therewith;

a second data source;

a second plurality of kernel mode data processing components including an originating component and a terminating component, the originating component reading data samples of a second data stream originating from the second data source, and at least one of the processing components synchronizing processing based on a master clock component associated with a processing component found in the first plurality of processing components; and

kernel mode component interconnections between the data processing components of respective first and second plurality of kernel mode data processing components to route the processing of respective data samples from the respective originating component to the respective terminating component.

36. A kernel mode media rendering system allowing synchronous processing of two media streams comprising:

a first media source;

a first plurality of kernel mode media processing components including an originating component and a terminating component;

the originating component reading media samples of a first media stream from the first media source;

the terminating component rendering said first media stream;

each media processing component having connection pin instances for passing first media samples between media processing components; and

at least one pin instance having a master clock component associated therewith;

a second media source;

a second plurality of kernel mode media processing components including an originating component and a terminating component;

the originating component reading media samples of a second media stream from the second media source;

the terminating component rendering said second media stream;

each media processing component having connection pin instances for passing second media samples between media processing components; and

at least one pin instance synchronizing processing of the second media stream with processing of the first media stream by using the master clock component associated with one of the pin instances of a processing component found in the first plurality of processing components; and

kernel mode component interconnections between the respective media processing components found in respective first and second plurality of kernel mode processing components created using the respective connection pin instances to route processing control of the respective media samples from the respective originating component to the respective terminating component.

3. Detailed Description of the Invention

1. The Field of the Invention

The field of the present invention is methods and computer program products relating to timing issues in multimedia applications, such as stream synchronization and rate matching between a generating clock and a rendering clock. The present invention also relates to standardized timing mechanisms in software drivers. More specifically, the present invention is directed to methods and computer program products for providing timing and clock mechanisms used by multiple entities, such as interconnected software drivers, in a standardized fashion so that multiple streams of processed data may be synchronized and rates between different hardware clocks may be matched.

2. Present State of the Art

When processing multimedia data, such as digitized sound and video data, it is common to process a continuous stream of media samples. The data stream also has time interval information associated with the data either by convention or by timestamp information. Timestamp information will tell a processing component when a particular sample is to be processed or can be used as a tracking reference to assure processing is progressing at the rate specified between timestamps.

Data organization, format, and convention can also convey time interval information used during processing. For example, a file of video data in a 30 frames per second (fps) format and convention will be understood by a processing component, having knowledge of the convention and format, that each sample is processed at 1/30 of a second intervals.

One problem encountered in processing media streams is that of synchronizing two or more streams so that they are rendered together. For example, an audio stream representing a soundtrack and an accompanying stream of video data need to be synchronized in order to produce a coherent multimedia presentation.

One method of synchronization utilizes a clock mechanism having a value that is based on a physical or hardware clock such as the PC clock. A data stream processor may query the value of this clock mechanism in order to synchronize processing or rendering of the stream. While synchronization occurs with respect to the hardware clock, it does not necessarily synchronize two different streams due to processing events and delays that may occur independently on each stream.

Another method of synchronization utilizes a clock mechanism that provides a time value that is based on presentation timestamp values on a "master" data stream while the stream is being processed. When the master stream is paused or otherwise stopped, the time value alternates to be based off a physical or hardware clock such as the PC clock. While the processor of a second data or media stream may query the value of this clock mechanism, further facility is provided so that the clock mechanism will notify the processor at a specific time on the physical clock or in certain intervals (*e.g.*, every 10 ms), again based on the physical clock.

Since event notification is based on the physical clock, such events may continue to occur even when processing is paused or stopped on the master data or media stream. Notifications may be temporarily disabled but at the cost of extra processing overhead and increased processing code complexity. Even when notifications are specifically disabled, timing circumstances may be such that a notification still is erroneously made.

Another problem associated with processing media or data streams is that of matching the rates of different hardware clocks associated with the media stream. One hardware clock will be used in the generation of the media stream media samples and making the timestamp information associated with each sample. A completely different hardware clock, however, will be used to render the media stream. Though the two hardware clocks may ostensibly run at the same rate, variations in physical clock rates may impact processing.

Rate matching, then, is the concept of compensating for actual frequency variations in physical clocks. For example, if a stream of live audio is being received through a network connection for rendering on local audio hardware, the rate at which the stream is being received may not match the rate at which the local audio hardware actually renders the stream. This is because the crystal on the audio hardware has nothing to do with the crystal used on the audio hardware on the other end of the network connection.

In some instances it is possible to perform fine adjustments on the local audio hardware in order to attempt to move it closer in frequency to what the source is producing. Not all audio hardware will allow this, however, and those that do have certain frequency resolution constraints that limit the accuracy of an adjustment and may have limits as to the amount of adjustment allowed.

Processing "drift" occurs due to this rate differential and ways are sought to determine the amount of drift accurately and quickly. Further, compensating for the drift optimally occurs in frequent minute increments, as opposed to infrequent gross adjustments, in order to avoid noticeable perception by an end user. Such perceptible adjustments come in the form of "skipping" of sound data and/or "jerkiness" of video images.

Yet another problem encountered with processing media or data streams is that of translating one time frame to another. This often occurs during synchronization as the presentation times of a "slave" stream are converted to the time frame of the master stream. Ways are sought to achieve such translations with minimal error introduction.

In a multimedia environment, it is advantageous to interconnect software drivers so that processing may occur in software drivers that run in a operating system mode with a much higher run priority and little security protection to allow access to actual hardware that the drivers, in many instances, directly manipulate. Many applications are benefited by running in this looser and more performance-oriented mode, generally referred throughout, in Windows NT terminology, as "kernel mode." Other robust operating systems will have

a functionally equivalent mode. Such interconnected software drivers have the same need for synchronizing multiple media streams and for rate matching of physical clocks.

One prime example of a program currently incapable of easily using kernel mode drivers, used throughout this application, comprises graph building functionality that allows a user to select and connect together different processing blocks, called filters, to successively manipulate a stream of multimedia data. The data typically is a series of samples representing sound or video and the processing blocks may include decompression processing for compressed data, special effects processing, CODEC functionality, rendering the data into analog signals, etc.

Such filters are typically located in user mode so that the graph builder portion of the program may interconnect and control their operation and be responsive to user input and rearrangement of processing blocks. Because of the consistent stream nature of multimedia data and the generation of large quantities of data, performance is a critical issue. In a general purpose operating system, system performance caused by repeatedly passing/switching back and forth between user mode and kernel mode can be so degraded as to prohibit certain multimedia applications.

Ways to standardize the interconnection of kernel mode filters would be desirable and would further benefit by a standardized clock mechanism to assist in solving the synchronization problem associated with processing multiple media streams simultaneously. Moreover, such standardized clock mechanism should also address and solve the rate matching problem associated with originating the data and rendering the data using different physical clocks.

SUMMARY AND OBJECTS OF THE INVENTION

It is an object of the present invention to synchronize processing of multiple data streams, each composed of data samples having time interval information associated therewith.

It is another object of the present invention to facilitate matching the processing rate of a data stream where the data stream was created by an originating physical clock as represented in the stream data samples and a rendering physical clock associated with a piece of hardware.

It is a further object of the present invention to provide a useful clock mechanism having a standardized interface for use in a system of interconnected kernel mode software drivers.

It is yet another object of the present invention to facilitate translation from time on one clock mechanism to time on another clock mechanism.

Additional objects and advantages of the invention will be set forth in the description which follows, and in part will be obvious from the description, or may be learned by the practice of the invention. The objects and advantages of the invention may be realized and obtained by means of the instruments and combinations particularly pointed out in the appended claims.

To achieve the foregoing objects, and in accordance with the invention as embodied and broadly described herein, a method and computer program product for synchronizing the processing of multiple data streams and matching disparate processing rates using a standardized clock mechanism is provided. The clock mechanism can be implemented in a variety of different ways and is explained throughout in the context of interconnectable kernel mode drivers providing efficient data stream processing.

The clock mechanism of the present invention maintains three different time values and makes them available to other components. The time values may be queried or the clock

mechanism may send notifications based on arriving at a given time value or at periodic intervals.

The first time value available at the clock mechanism of the present invention is a positional time value that is based on the time interval information associated with a data stream and reflects the "position" within the data stream being processed. The second is a physical time value based on an actual hardware oscillator or clock, such as the PC clock or a rendering processor clock. Finally, the third is a correlated time value that provides a designated time value, such as the positional time value, together with a reference time value based on a commonly available hardware clock such as the PC clock.

With the clock mechanism as a master clock, the positional time value can be used by other processing components processing other data streams for synchronization. Since the positional time value is based on processing of a data stream, when the processing of the underlying data stream stops the clock does as well. Furthermore, timing event notifications will also be frozen on an absolute time basis, but will continue to be perfectly timed with respect to the positional time value. This provides a great benefit in that "paused" time need not be tracked, manipulated, or otherwise be of concern. "Slave" clock mechanisms and components will base processing on time as it is provided by the positional time value.

The physical time value of the underlying hardware clock associated with a data stream renderer or other processor can be used to match the rate of the data stream being processed with the processing rate of the rendering processor. Since the data stream has time interval information associated therewith, the originating processor's processing rate is inherent therein and may be ascertained and compared to the rendering processors actual processing rate as found in the physical time value. This allows any processing rate "drift" to be quickly and accurately determined thereby allowing processing adjustments to be made with less noticeable effect on the rendered data.

Finally, the correlated time value provides a means for other clock mechanisms and processing components to translate time information from one basis to another. The common hardware clock must be available to other clock mechanisms or components so that it may be used as a common reference. Since the correlated time is returned or accessed as an atomic operation, any errors introduced are minimal and non-cumulative.

Two variations of correlated time are supported in the described embodiment, both using the PC clock as a reference. One uses media time as the designated time and the other uses the physical time as the designated time. Throughout this application, correlated time used alone could refer to either of the two variations or even other variations that will be apparent to those skilled in the art. When distinguishing between the variations, the terms correlated media time and correlated physical time will be used with the respective designated time compared to the common reference.

One embodiment of the present invention occurs as part of connecting kernel mode drivers in a standardized fashion. A given driver or filter will support and define connection point "pin factories" that may be used to instantiate connection pin instances that are interconnected to other pin instances on other drivers to allow processing messages to be consecutively processed in kernel mode by the drivers without necessary resort to a user mode agent.

A third party agent desiring to connect compliant drivers will query the drivers of their capabilities. Such capabilities include what kinds of connection pin instances may be instantiated, including their relevant characteristics, such as type of data handled, data formats, transfer rates, medium or mode of transfer, input or output nature of a connection pin instance, etc. Also queried will be the data buffering requirements and capabilities for allocating buffers available at each connection pin factory.

Once a third party agent, typically running in user mode, has queried the capabilities of one or more compliant drivers, the agent will determine the best connection characteristics

for "chaining" multiple drivers together so that data may be optimally processed between them. This determination step occurs after all driver capabilities have been queried so that the optimal connection criteria may be selected.

For each connection pin instance, the third party agent will also determine whether a buffer allocator need be created on a particular pin instance. Again, this is done after having queried all the different filters and pin factories prior to making any interconnections.

The third party agent then interconnects the drivers by creating a connection pin instance based on the connection pin factories on each driver. The agent will specify a data format and a connection format as part of the connection pin instance creation. In an exemplary embodiment implemented under the NT operating system, an actual connection pin instance is created by a create I/O operation that returns a handle to a "file." The create I/O request will contain the driver instance handle and reference to a data structure indicating data format and connection format for the connection pin instance.

Furthermore, reference to previously created connection pin instances (*e.g.*, input pins or IRP "sink" pins) will be specified in requests for creating other connection pin instances (*e.g.*, output pins or IRP "source" pins) in order to effectuate a connection between connection pin instances. After a source pin instance is created using a reference to an input pin instance having a buffer allocator, indication is made with the source pin instance of the buffer allocator so that data may be transmitted into the new buffer from the existing buffer. If no reference is indicated, the source pin instance will leave the data in the existing data after processing.

In order to create a compliant driver, a driver developer will support certain standard facilities to allow a user mode agent to query capabilities and make interconnections between drivers. In one embodiment, built on the Windows NT operating system, this is achieved by use of "sets" (*i.e.*, property, method, and event sets) that implement the desired functionality.

A set is logically defined as having a GUID (globally unique identifier) to identify the set as a whole and a RUID (relatively unique identifier, *e.g.*, relative within the set itself) for each element of functionality within the set. Each set is associated with only one or two IOCTLs (IO Controls), and an IOCTL combined with a set specification controls all interaction with the driver.

As currently embodied, three types of sets are utilized, namely, property sets, method sets, and event sets. Property sets are used for managing values or settings within the driver, such as sound volume, transfer rate, etc., and use a single IOCTL with a flag indicating whether the call is getting a property value and or setting a property value. Method sets are used for managing the operations that a driver may perform, such as allocating memory, flushing buffers, etc., and uses a single IOCTL to call the specified method. Event sets are used for managing events associated with driver processing, such as device change notification, data starvation notification, etc., and uses two IOCTLs, one for enabling a specified event and one for disabling a specified event.

To use a set, an I/O control operation is initiated using the specified IOCTL and reference to a data structure having the set GUID, RUID, and other necessary data. For example, setting a volume property on a sound card driver would entail an I/O control operation using a set property IOCTL, specifying the appropriate GUID for the property set having the volume setting, indicating the specific RUID within that set indicates the volume property, and containing the new volume setting value.

To query the sets supported, a null GUID is used along with a query flag on a specified IOCTL for a particular set type (*e.g.*, property set IOCTL, method IOCTL, or event enable IOCTL) and a list of set GUIDs supported will be returned. To query supported properties, methods, or events within a given set, the set GUID, set type IOCTL, and a query flag are used with the operation returning a list of supported RUIDs.

By using the generic set mechanism, a minimum of functionality may be implemented to support a compliant driver but still allow unlimited extensibility. A set may be defined in a written specification that can be independently coded by a multitude of different driver developers to create a system of interoperable and interconnectable drivers as long as particular sets are implemented. Furthermore, the specification can define mandatory properties, methods, and events that must be supported as well as optional properties, methods, and events that can be implemented depending on the driver functions and advanced capabilities. In addition to the basic minimum commonality required, driver developers may incorporate additional functionality by defining their own sets and assigning them a GUID. By being able to enumerate supported functionality (*i.e.*, make queries for supported GUIDs and RUIDs), a caller, such as a third party controlling agent, can adjust expectations or make appropriate compensation depending on the capabilities of the underlying filters.

An embodiment of a clock mechanism having a positional time value, a physical time value, a correlated positional time value, and a correlated physical time value may be created on a compliant connection pin instance. A "file" is opened on the particular device in the I/O indicating the handle of the file object representing the connection pin instance as parent and a GUID string value indicating the clock mechanism file type. A handle is returned to a file object representing the clock mechanism and the filter will support, through this file object, property sets for providing the respective time values. The positional time value will be based on the data stream processed by the filter and passing through the particular connection pin instance. The physical time will be based on a hardware clock or oscillator associated with the filter, if present, or the PC clock. Both of the correlated time values will typically use the PC clock as the reference value due to its widespread availability.

The clock mechanism may be used as a "master" allowing other clock mechanisms or components to "slave" off or synchronized with it's time tracking. Furthermore, a default implementation based on the PC clock is provided.

These and other objects and features of the present invention will become more fully apparent from the following description and appended claims, or may be learned by the practice of the invention as set forth hereinafter.

In order that the manner in which the above-recited and other advantages and objects of the invention are obtained, a more particular description of the invention briefly described above will be rendered by reference to specific embodiments thereof which are illustrated in the appended drawings. Understanding that these drawings depict only typical embodiments of the invention and are not therefore to be considered limiting of its scope, the invention will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

As used herein, the term "user mode" refers to a level of operation in an operating system where most user written programs run. The user mode level of operation is typically the most secure level and has a significant amount of overhead to prevent one application program or process from interfering with another application program or process. Furthermore, access to system resources is highly controlled through specific interfaces and run priority is generally one of the lowest, if not the lowest.

As used herein, the term "kernel mode" refers to a level of operation in an operating system having significantly less restrictions than the user mode level of operation. Examples of kernel mode programs or processes would include software drivers for controlling hardware components. Typically, kernel mode programs are performance sensitive, and therefore, have less operational overhead than user mode programs. Furthermore, access to hardware and many system resources is unrestricted or much less restricted than for user mode programs. In many instances, program code running in kernel mode relies on programmer discipline and conformity to convention in order to establish good system behavior (*e.g.*, not disrupting another program's address space, etc.). Another term used for kernel mode is "trusted" code.

As used herein the term "driver" refers to software driver programs typically running in kernel mode. The term driver may also refer to the actual executable program that is loaded onto the operating system or a portion thereof that imparts certain functionality. Drivers are in many instances, though not necessarily, associated with some form of hardware.

As used herein, the term "filter" refers to a portion of the functionality found within a software driver, including the entire driver itself, where connection points may be exposed for sending data through the filter. For example, a software driver may support a number of different filters or may have one single function. Furthermore, a number of filters from

different drivers that are internally connected together and externally exposing connection points for input and output may collectively be referred to as a single filter. Also, in a more generic sense, the term filter may refer to the operation performed, such as decompression, etc., regardless of whether that occurs in a software driver filter running in kernel mode or another piece of program code running in user mode.

As used herein, the term "driver object" refers to an operating system entity, defined by the operating system, for managing and making known a software driver as a system resource.

As used herein, the term "device object" refers to a system level entity defined by the system, for making known a portion of a drivers functionality available for use as a system resource and defines the driver functionality and availability to other system components. Both the driver objects and device objects are typically created upon loading and initialization of the driver software.

As used herein, the term "file object" refers to an operating system entity, defined by the system, for managing an invocation of a resource specified by a device object. A file object provides a context for usage of a driver object. Furthermore, a file object may be hierarchically related to another file object if the previous file object is designated as a "parent" during the creation of the new file object. File objects are typically used in managing all I/O operations that operate on a stream of data.

As used herein, the term "data" refers to any information that is processed through the interconnected kernel mode filters. Such data includes media data representing video, audio, text, MIDI, etc. but may also include control information or parameters for other applications. For example, a kernel mode filter graph may be used in process control operations where the control information passed between the different filters is used to develop control signals for actuating machinery. While examples are given of media

processing systems, other applications could in like manner benefit from the system of interconnected kernel mode filters explained herein.

Throughout this application, the description of the present invention is described within the context of the Windows NT™ operating system available from Microsoft®. Furthermore, familiarity with the Windows NT I/O architecture is presumed in order to understand the preferred embodiment explained herein. A good tutorial of the I/O system as well as the NT operating system in general can be found in the book "Inside Windows NT" written by Helen Custer and published by Microsoft Press and is further incorporated herein by reference.

While the discussion of the drivers and system entities such as file objects, device objects and driver objects are explained herein within the context of how they operate in the Windows NT operating system, those skilled in the art will appreciate that the present invention may be implemented on other operating systems having analogous components, whether or not they use the same terminology. For example, should another operating system have an entity that operates as a file object, it could be interpreted as a file object regardless of its actual title.

Referring now to Figure 1, an example system is presented for reading a stream of sound data from a disk drive and rendering that sound data so that it may be heard through a speaker according to the prior art model. An amount of data is stored on hard drive 20 representing sound in the form of digitized sound samples. Alternatively, the source of the sound data stream may be digitized information coming over a phone line, digitized information from network or other communication packets, or other sources known in the art. The data stream is composed of digitized samples having time interval information associated therewith either by data format and convention or by explicit timestamp information attached to each sample. A kernel mode disk driver 22 interacts with the disk drive hardware 20 and is under control of a user mode reader program component 24. A

controlling agent 26 manages the different components in order to effectuate the rendering of the sound data and may include dynamic graph building capabilities so that the different software components may be dynamically allocated in order to provide custom filtering or other processing paths as designated by an end user.

The reader component 24 will interact with disk driver 22 using a standard I/O control interface of the operating system and will cause the compressed sound data to be read from the disk drive 20 into buffers allocated in user mode as part of the user mode process address space. Next, a decompressor component 28 will decompress the compressed data into its decompressed format for processing. As shown, this step is done entirely in user mode with the attendant lower priority and process behavior safety mechanisms.

The effects filter 30 will operate on the data to provide some special effect and will have an accompanying effects filter 32 operating in kernel mode. Furthermore, an effects processor 34 may be present or the effects filter may operate entirely in software emulating the actual hardware processor. In order to access the effects filter 32 the effects component 30 will use the system I/O control mechanism to transfer the data and control to the effects filter. Again, the kernel mode/user mode boundary is crossed in order to make this transition.

The effects filter 32 will control the effects processor 34 and cause whatever special effect is necessary or desired to be made on the data. This may entail copying all the data from the effects component 30 down to the effects filter 32 and again to the effects processor 34 depending on actual system configuration. While many software effects components have a hardware processor associated therewith, others function entirely within system software running on the host processor.

After control and the data is transferred back into user mode at the completion of the processing of the effects component 30, it is then transferred to sound rendering component 36. The sound rendering component 36 will transfer the control and data to the

sound rendering driver 38 which in turn controls the sound card 40 in order to render the data, as processed and filtered, as sound through speaker 42. As can be readily seen, there exists a variety of transfers between user mode and kernel mode that add inefficiencies to the rendering of the sound data. Because of the timing sensitive nature of multimedia data, such as a continuous stream of sound or video, it is advantageous to reduce these inefficiencies and transitions of control as well as the multiple copying of data between different buffers.

One embodiment of the present invention and used throughout will consist of a service provided on the Windows NT operating system architecture. This service is broken into different software components that a user of the system will access. First, a user mode API is available that will include a routine for creating connection pin instances and other file objects representing particular functionality, such as a clock mechanism or a buffer allocation mechanism. Additionally, and more importantly, there will be a complete set of routines and data structures to assist the driver developer in making drivers that are compliant with the standardized architecture. By utilizing such facilities from the system, different driver developers may create compliant drivers that will interact with one another according to the specified architecture. User mode agents communicate with compliant drivers through an environment subsystem running in user mode that will communicate with the system services of the NT executive and the I/O manager. This is the same standard I/O mechanism for all other I/O and the present implementation of the preferred embodiment will utilize existing system services as much as possible.

The architecture of the system of Figure 1 utilizing the present invention will appear as shown in Figure 2. A controlling agent 44 will query the drivers known in order to then make interconnections according to data format and connection format to effectuate the rendering entirely in kernel mode. Furthermore, the controlling agent will receive notifications of important events so that it may exercise control as necessary. Examples of such events would include end of processing, a data starvation situation, etc.

In this configuration, the sound data is read from disk drive 46 by the disk driver 48, as before. A reader driver 50 controls disk driver 48 and is "vertically" associated with disk driver 48 according to the NT layered I/O architecture as used in conventional fashion. The terms vertically and horizontally are used to distinguish driver connections that currently occur as part of the NT layered I/O architecture (vertical) and connections according to the interconnected kernel mode drivers made dynamically by a third party controlling agent (horizontal).

Reader driver 50 is also interconnected "horizontally" to a decompressor driver 52 according to the connection methods explained hereafter and is managed by the controlling agent 44. Decompressor 52 will perform the decompression in kernel mode before passing the data and control to the effects filter 54. The effects filter will apply the special effects utilizing an effects processor 56 as necessary before passing the data and control to the sound rendering driver 58 that controls the sound card and causes the data to be rendered as sound on speaker 62. As can be noted by reference to Figure 2, keeping processing in kernel mode represents an efficiency advantage by eliminating multiple transitions between user mode and kernel mode and by reducing the amount of overhead normally associated with processing in user mode.

Referring now to Figure 3, a logical diagram showing the hierarchal nature of system objects related to interconnected software drivers compliant with one embodiment of the present invention is shown. A driver object 64 is created to represent the executable software code image as loaded in memory. The driver code image contains the entirety of the driver's functionality, and the driver object 64 includes information regarding the image, such as its location on the system, the kinds of devices supported, etc.

For each type of independently accessible functionality by a controlling agent, device objects 66a-66N are created in the I/O directory structure representing the different functions that are available that will be accessed by user mode clients. These typically

represent filters or other portions of functionality independently available. The driver object 64 and the device objects 66a-66N are created upon installation of the driver code as represented by the enclosing box 68.

Historically, a device object exists for each element of physical hardware. The flexibility in modern I/O systems, however, allows a device object to represent a filter implemented entirely in software. As such, device objects may be readily created for each instance of a filter implemented solely in software. A software filter may therefore be implemented so that each instance as represented by a device object has a one-to-one correspondence with a device object or a single device object may follow the more traditional approach and manage multiple file objects, with each file object representing a client instance of the filter.

Upon a device object, as shown for device object 66a, file objects are created representing independent instances of the functionality represented by device object. While a device object represents a filter and may manage multiple instances of that filter, a file object represents the actual instance of that filter used by a particular entity. Therefore, file object 70 is an instance of the filter defined by device object 66a.

To use a filter, the controlling agent or other user mode client opens a file on a device available in the I/O directory structure. A file object with appropriate context information will be created and a handle to that file object returned to the user mode client. While file objects may be hierarchally related by specifying a "parent" file object during creation, file objects will also have a sibling relationship in that they are all children of the same device object.

Context information within a file object consists of information to manage the I/O interface with user mode clients, the "state" of the entity that the file object represents, etc. The context information has system required information and further includes user definable areas that can be given special meaning. An example of how the user definable area can be

used will be shown hereafter discussing the implementation of a validation and IRP routing method.

In order to provide connection pin instances, the file object 70 representing a filter instance will be used as a parent in creating children file objects representing the connection pin instances for a particular filter. While file object 70 will be queried for the connection pin factory definitions and availability, actual file objects will be created for each instance of such a pin factory, using the particular file object as the appropriate informational context in order to validly and correctly create the connection pin instance. For example, file objects 72 and 74 represent connection pin instances for the filter represented by file object 70 and are hierarchally related to file object 70. The connection pin instances, as represented by file object 72 and 74, respectively, may be a data path into and then out of the filter instance (represented by file object 70) which can be used for connecting to other connection pin instances in forming a series of chained filters or other driver functionality.

Just as a pin instance is represented by a file object having a hierarchial relationship to another file object representing the filter instance in order to provide context information for the pin instance, other file objects may be hierarchically related to a pin instance in order to represent other functionality so that proper context information is available. Context information is necessary to distinguish one pin instance from another according to the individual parameters used in creation, such as pin data format, communication type, etc.

Other operational entities, such as a buffer allocation mechanism, a timing mechanism, etc., requiring either an individual context or user mode control through a handle may also be represented by file objects. Furthermore, hierarchical relationships between the file objects (*e.g.*, a buffer allocation mechanism associated with a particular connection pin instance) may be established if necessary by specifying a parent file object during creation of the child file object. These parent/child relationships exist to determine relationship and structure between the file objects representing the operational entities. Additionally, a

particular type of "parent" file object will only be able to produce certain types of "children" file objects, thus requiring the creation validation mechanisms as explained hereafter. Again, such file objects have corresponding handles available in user mode that are returned to a client through a system API call such as NtCreateFile.

The handles to file objects are used by user mode clients, such as a controlling agent, to communicate with the kernel mode drivers. The hierarchical chain of file objects, device objects, and driver objects allows the I/O subsystem to traverse back to the driver object through the hierarchically related file objects and device objects to arrive at the entry points into the actual driver code. Such entry points are references (*e.g.*, pointers) to functions in the software driver code. Furthermore, each of the objects in the object pathway between a particular file object and the driver object having the entry points to the software driver code provides important context information for the I/O subsystem in creating IRPs as well references into data structures used for properly routing IRPs according to the routing and validation mechanism explained hereafter.

Handles for file objects and other system objects are process-specific and are the means by which a user mode process will communicate with an underlying object. It is interesting to note that multiple handles may be created to reference a single underlying system object, such as a file object. This means that multiple applications may be feeding information to a pin instance as represented by a file object.

One element of information that is important for interconnecting different drivers is the device object stack depth parameter. This will indicate the IRP stack location of a particular driver object. In this manner, a single IRP may be used and passed between interconnected drivers using the I/O manager, thereby providing a performance enhancement over separately creating and sending IRPs between the various interconnected drivers. Alternatively, each driver could create through appropriate I/O manager calls new IRPs for

each successive communication and cause each new IRP to be sent to the next driver in the chain of interconnected drivers.

Referring now to Figures 4A-4C, extensions to the system driver objects, device objects, and file objects are shown that allow validation of file object creation of differing types as well as I/O Request Packet (IRP) routing to appropriate handlers. Figure 4A shows a driver object 76 representing the executable code implementing one or more filters or other driver functionality. Within the driver object, the Windows NT architecture requires a reference to a create handler provided by the software driver developer. According to this embodiment, a multiplexing dispatch function 78 is referenced from the driver object 76 as the create handler and will be used to route messages to particular create handlers depending on the file object type to be created. Operation of the multiplexing dispatch function 78 will be explained in connection with the flow chart shown in Figure 6 hereinafter.

In like manner, other handlers from the driver object will indicate a multiplexing dispatch function and, depending on implementation, they may be the same function. In other words, as explained in more detail below, each type of I/O handler reference (*e.g.*, read, write, device control, etc.) will point to a multiplexing dispatch function that uses the extension data in a device object and the context information in a file object in order to route the message to the appropriate handler. The extension data in the device object that references a validation table will be used when no parent file object is specified on a create operation. Otherwise, the parent file object context information will indicate the correct validation table.

Figure 4B shows a driver object 80 which has a particular device extension area 82 that can be utilized as desired by the driver developer and includes driver specific information. At a defined location within the device extension area 82 of the driver object 80 is a reference to a data structure, known as a file type validation table 84, containing string representations of file object types 86 and references to the associated create handlers 88 for

each file type represented. The create multiplexing dispatch function will the utilize file type validation table 84 to validate the file object type to be created and then turn control over to the appropriate create handler as will be explained in detail hereafter in connection with the discussion of Figure 6. The string to be validated is found in the IRP create request and originates from the file name string used with the NtCreateFile function call in user mode. The NtCreateFile call is made within the user mode function cell to create a pin instance or other mechanism.

Figure 4C shows a file object 90 having a file context area 92 that is free to be used by the software driver developer. Reference is made from the file context area 92 to an IRP request handler table 94. The different types of IRP request 96 are associated with references to particular handlers 98, and the appropriate multiplexing dispatch function will use this information to access the correct handler. In the case of determining the correct create handler, a data structure known as a file type validation table 100 is referenced having string representations of file object types 102 and references 104 to the associated create handlers for each file type represented. For children file objects (*i.e.*, file objects that have another file object rather than a device object as parent), the type is represented by a string that is compared to the strings in the file object types 102. When a match is found, the associated create handler is accessed using a reference from the references 104 that is associated with the matched file object type string. If no match is found, then the request is invalid and an error indication made.

Referring now to Figure 5, the installation procedure for setting up the creation validation and mechanism is shown. At step 106, the installing program will make reference in the driver object to the appropriate multiplexing dispatch functions. As shown in Figure 4A, the create handler points to a generic mulitplexing dispatch function. In like manner, all other handler references in the driver object 76 would point to other generic multiplexing dispatch functions germane to the particular handler as necessary. Alternatively, each

handler reference could point to the same multiplexing dispatch function that could in turn process the IRP request and route it to the appropriate handler. Such an alternative multiplexing function will necessarily be more complex in order to accommodate different kinds of request (*e.g.*, create, write, etc.).

Next, at step 108, each device object created as part of the software driver executable code installation will be adjusted to reference the file type validation table 84 as shown in Figure 4B. Finally, at step 110, the processing of IRP requests will begin with the multiplexing dispatch function using the file type validation table 84 as referenced from the appropriate device object 80

When a file object is created, the appropriate IRP dispatch table 94 will be created and referenced along with the indexed file object type validation table 100 as necessary. The creation of the file object type validation tables occurs within the provided create handlers according to file object type. The data structures are created representing the IRP dispatch table 94 and the file object type validation table 100 and a reference thereto stored at a specific location with the file context information 92 of the particular file object 90 being created.

Referring now to Figure 6, a flow chart is presented showing the operation of the create multiplexing dispatch function and its validation mechanism including its interaction with the data structures referenced from the system driver objects, device objects, and file objects. At step 112, a user mode process sends an I/O request for creating a file object. This I/O create request is made using an invocation to the system API for NtCreateFile. At step 114, the I/O manager sends the IRP to the multiplexing dispatch function 78 based on the reference in the driver object 76 (*see* Figure 4A).

Once the multiplexing dispatch function 78 has the IRP for the create request, a test is made at step 116 to determine if there is a parent file object. The information necessary to make this determination will be found within the IRP itself and originally be supplied by

the user mode process. The user mode process will supply a handle referencing the "parent" file object as part of the create request and the NT system will create the IRP having the correct reference to the "parent" file object.

If there is no parent file object, the right branch is taken, and the multiplexing dispatch function 78 uses the device extension 82 from the appropriate device object 80 to reference a file type validation table 84 (*see* Figure 4B) at step 118. Using the validation table 84, the multiplexing dispatch function 78 will validate the file object type at step 120 by comparing the string in the request with the file object types 86 strings.

If there is a matching string as determined at step 122, the appropriate create handler is accessed at step 124. Otherwise the create request is rejected at step 126. The create handler as accessed at step 124 will create, or cause to be created, the file object at step 126. With the created file object, the appropriate create handle will make the file object reference in the file context 92 to an IRP dispatch table 94 that it has previously created.

Again at step 116, it may be determined that there is a parent file object present. If a parent file object is present, as determined at step 116 as found in the IRP associated with the create request, the multiplexing dispatch function 78 uses the file context 92 from the parent file object 90 to reference an IRP dispatch table 94 (*see* Figure 4C) at step 130. For a create request, the multiplexing dispatch function 78 will access a file type validation table 100, at step 132. Using the file type validation table 100, the multiplexing dispatch function 78 will validate the file object type at step 133 by comparing the string in the request with the file object types 102 strings, as was done above.

If there is a matching string as determined at step 134, the appropriate create handler is accessed at step 138. Otherwise the create request is rejected at step 136. With the appropriate create handler, the file object is created at 140, and the create handler will make a new IRP dispatch table 94 for the newly created file object and will make reference in the newly created file object 90 file context area 92 to the newly created IRP dispatch table 94

at step 142. Note that the same file object structure as shown in Figure 4C is used to explain interaction with both the parent file object and the validly created child file object. While the same structure exists in both cases (once the new file object is created), they will be used differently and contain different information.

Whenever a connection pin instance is created, a connection pin ID is passed that identifies the pin factory in the filter that "supports" the creation of the pin instance. Those skilled in the art will note that the connection pin ID may also be validated as a string in a validation table in much the same manner as the file object is validated and that other implementation variations exist.

In order to make connections between different drivers, a common mechanism must be present to assure that a given driver supports such interconnections. This common mechanism must allow discovery of filter capabilities including connection pin factory capabilities. Furthermore, such a mechanism should also be extensible to provide additional flexibility to driver developers.

One mechanism chosen in the present embodiment for defining compliant drivers and allowing discovery of capabilities are identified "sets" of related items. This is a convenient mechanism to be used with existing I/O communication mechanisms. A set is logically defined as having a GUID (globally unique identifier) to identify the set as a whole and a RUID (relatively unique identifier, *e.g.*, relative within the set itself) for each element of functionality within the set. The set identifier and any other data structures necessary for operating with the chosen RUID item are passed as part of an I/O control call using the filter handle as a parameter. Only a small number of IOCTLs need to be allocated in order to implement a full system of functionality. As implemented, three different types of sets are established depending on their functions, requiring a total of four IOCTLs. Other implementations may use sets in a different manner. The particular IOCTL will signal the handler for I/O control to interpret or use the chosen element (using the RUID) in a certain

manner. Furthermore, control flags may be passed with the GUID and RUID to further specify control information.

The first set type is a property set and is used in connection with values or settings found within the driver or on any associated hardware. Examples of such settings would be transfer rate, volume level, etc. One IOCTL is associated with property sets with a control flag differentiating between a "get" property and a "set" property command. In this manner the same data structure can be used to either set or get a particular property with the driver determining the action required based on the IOCTL used. The correct property is identified by the set identifier consisting of its unique GUID and RUID combination.

Method sets are another type of set used and are a set of actions that can be performed by a driver. Only one IOCTL is needed to identify the method set with the correct method to be actuated identified by the unique GUID and RUID combination for the set identifier. Methods are used to control the driver and include such functions as initializing the driver for use, clearing buffers, etc.

Event sets are used for managing events associated with driver processing, such as device change notification, data starvation notification, etc., or any other notification defined by the set that may be useful to a user mode application. Two IOCTLs are used, one for enabling a specified event and one for disabling a specified event, while any data structures necessary for a given event identified by a RUID can be shared whether enabling or disabling the event.

To use a set, an I/O control operation is initiated using the specified IOCTL and reference to a data structure having the set GUID, RUID, and other necessary data (*e.g.*, control flags, data structures, etc.). For example, setting a volume property on a sound card driver would entail an I/O control operation using a property set IOCTL, a control flag indicating a set property operation, the appropriate GUID for the property set having the

volume setting, the specific RUID within that set indicates the volume property, and the new volume setting value.

To query the sets supported, by type, an IOCTL for a particular set type (*e.g.*, property IOCTL, method IOCTL, or event enable IOCTL) having a null GUID and control flags to indicate supported set enumeration are issued as part of an I/O command and a list of set GUIDs supported will be returned. To query supported properties, methods, or events within a given set, the set GUID, set type IOCTL, a null RUID, and control flags indicating enumeration of supported elements are used with the I/O operation. A list of supported RUIDs will be returned as a result of the I/O operation. This will allow a third party agent to determine which, if any, optional elements of an implemented set are supported.

The written specification of a set uniquely identified by a GUID allows a documented mechanism that both driver developers and third party controlling agents may use as an implementation guide. The third party developer will know of a given driver's capabilities based on response to queries and preprogrammed knowledge based on the abstract set definition. Likewise, a driver developer may use the abstract set definition as a guide to implementing a set or group of sets providing known functionality to any third party agent.

In order to provide the connection abilities described herein, a compliant driver must support certain sets. The following tables illustrate some important kinds of information that may be supported in property set format and that can be used in implementing the present invention. The first table refers to properties about a connection pin factory that would be implemented by a filter, while the second table refers to properties about an actual connection pin instance that would be created by using a particular connection pin factory as a template.

TABLE 1

Filter Properties and Their Use	
Property	**Description**
Connection Pin Factories	Lists the different types of connection pin instances that may be created on a particular filter, each distinguishable type referred to as a pin factory. Note that this is not the total number of connection pin instances which could be instantiated on this device, but the number of unique connection pin types, such as an audio input and audio output.
Connection Instances	Lists the number of instances already created of a given connection pin factory as well as the highest number of instances supported for that particular connection pin factory. If the total cannot be determined until the filter is actually connected, this property will return a -1.
Data Flow	Lists the possible data flow direction of a connection pin factory with respect to a filter (*e.g.*, into the filter, out of the filter, or either into or out of the filter).

Communication	Lists the communication requirements for a given connection pin factory in terms of processing IRPs. Some connection pin factories may not be interconnected but have other forms of control mechanisms associated therewith such as a "bridge" to a file source for data that represents a source point on a graph. The bridge control mechanism would allow setting of a filename indirectly where information is stored. In an exemplary implementation, an agent (which decides which pin factory to use for making a connection pin instance) must be able to understand the intrinsic meaning of a "none", "sink" or input, "source" or output, "both," and "bridge" communication types for a connection pin factory. For example, a source connection pin instance requires a handle or reference to a sink connection pin instance, etc. In the communication type context, sink and source refer to the disposition of the connection pin instance in processing IRPs. A sink would receive the IRPs for processing, while a source would pass the IRPs onto the next appropriate processing component. There are two communication types that are neither sink nor source and represent end points in the connection graph. An end point represents the place where data either enters or exits from the connected filters. A none designation indicates that the connection type may not be instantiated while a bridge communications type refers to an end point that may be instantiated so that specific properties may be manipulated. For example, a bridge end point that is part of a file reader will likely have a property that will contain the path and file name of a file that stores the data to be processed.
Data Ranges	Lists the possible data ranges that a connection pin factory may support, including the format of the data, if relevant. In one implementation, a count followed by an array of data ranges, which the connection pin type may support, is used as part of the property. In that implementation, if different data ranges are supported under different mediums or interfaces (see below), different connection pin factories are available on a particular filter to accommodate such differences. Furthermore, each data range structure may be extended for format specific detail such as number of bits and channels. The actual data format a connection pin instance uses is set during creation of the instance. The data range property is used to assist in determining what that actual data format should be for a particular connection pin instance and is accessed or queried by a third party controlling agent.

Interfaces	Lists other set GUIDs indicating the supported interfaces on a particular connection pin factory. An interface is the type or types of data that may be communicated through a connection pin factory. For example, MIDI data, CD music, MPEG video, etc., would be interfaces in the sense that data has a particular convention and format that a filter could handle. Such interfaces also comprise protocols for submitting the data. An interface is independent of the medium by which it is communicated.
Mediums	Lists the supported mediums on a particular connection pin factory. A medium is the way or mechanism by which information is transferred, such as IRP-based, sockets, etc. An interface may be defined on top of a variety of different mediums. In the preferred embodiment and implementation explained herein, an IRP-based medium and file IO- based medium is used.
Data Intersection	Returns the first acceptable or "best" data format produced by a connection pin factory given a list of data ranges. This approach is used to allow a third party agent to determine data requirements when chaining different filters together. In one implementation, the data intersection property is used to determine the best data format produced by a connection pin factory given the constraint of a list of data ranges. The list of data ranges may be acquired using the data ranges property on another pin factory that will be connected as explained previously. A third party controlling agent, which has no knowledge of the data type specifics, may use the data range list of one connection pin factory and return the "best" (e.g., first acceptable data format) data format on the current connection pin factory. Although a set of ranges of the two intersecting connection pin factories could be returned, only the best format is returned by the driver. In this manner, the third party controlling agent can apply this "best" data format to the next driver in the graph in order to create a virtual set of connections before actually initiating the creation of connection pin instances and connecting the entire graph of filters together. This allows the controlling agent to assess the viability of a particular filter graph selected by a user and point out potential problems to the user before actually connecting the graph. The data format returned can also be restricted by the formats available given the connections already made on the filter. This property is capable of returning an error if a particular data format cannot be determined until an actual connection is made or if an intersection is dependent on multiple data formats on different connection points. Essentially, intersection information is provided while the property itself will return a data format.

TABLE 2

Connection Pin Instance Properties and Their Use	
Property	**Description**
State	Describes the current state of the connection pin instance. Possible states include being stopped, acquiring data, processing data, being paused or idle, etc. The state represents the current mode of the connection pin instance, and determines the current capabilities and resource usage of the driver. The stop state is the initial state of the connection pin instance, and represents the mode of least resource usage. The stop state also represents a point wherein there will be the most latency in data processing in order to arrive at the run state. The acquire state represents the mode at which resources are acquired (such as buffer allocators) though no data may be transferred in this state. The pause state represents the mode of most resource usage and a correspondingly low processing latency to arrive at a run state. Data may be transferred or "prerolled" in this state, though this is not actually a run state. The run state represents a mode where data is actually consumed or produced (*i.e.*, transferred and processed) at a connection pin instance. More resolution in control may be accomplished using custom properties depending on the purpose of the filter and the underlying hardware. For example, in order to make an external laser disk player spin up, one would set some sort of custom "mode" property specific to that class. Setting this property may also change the state of the device but not necessarily, depending on the effect of the mode.
Priority	Describes the priority of the connection pin instance for receiving access to resources managed by the filter and is used by the filter in resource allocation arbitration. This property, if supported, allows a third party controlling agent to indicate the priority placement of the particular pin instance relative to all other connection pin instances of all other drivers which may share limited resources with this particular connection and instance. This priority property may also be implemented to allow an agent to set finer tuning of the priority within a single class of priority. For example, a priority may have certain subclasses associated therewith. If two drivers competing for the same resources have the same priority class, then the subclass priority is used to determine resource allocation between the two drivers. If the subclass priority is also the same, then arbitrarily, the first connection pin instance will receive the resources.
Data Format	Used to get or set the data format for the connection pin instance.

The previous tables are given by way of example and not by limitation, and those skilled in the art will appreciate that many different properties and schemes may be implemented in order to create the connections between different drivers. One important element is the standardization factor so that different driver manufacturers or development groups may create drivers that may be interconnected since they are able to implement the same property sets.

Another useful property set gives topology information for the internal relationships of input and output connection pin factories on a given filter. This information will state the relationship of input pin factories and corresponding output pin factories on a given filter as well as what type of processing happens between the input and output pin factories. Examples of the processing that occurs would be different data transformations, data decompression, echo cancellation, etc. Such information is useful to an automated filter graph builder that will trace a hypothetical connection path using multiple filters before making actual connection pin instances and connections. Essentially, the topology information explains the internal structure of the filter and exposes this through a property set mechanism to inquiries from third party agents.

Therefore, a compliant driver is simply one that implements the designated property set. This allows a third party controlling agent to make queries and settings to the compliant filter once it is determined that a given property set is supported. The overall goal is to acquire enough information on how to connect the differing filters together in order to form a filter graph.

By using the generic set mechanism, a minimum of functionality may be implemented to support a compliant driver but still allow unlimited extensibility. A set may be defined in a written specification that can be independently coded by a multitude of different driver developers to create a system of interoperable and interconnectable drivers as long as particular sets are implemented. Furthermore, the specification can define

mandatory properties, methods, and events that must be supported as well as optional properties, methods, and events that can be implemented depending on the driver functions and advanced capabilities. Besides the basic minimum commonality required, driver developers may incorporate additional functionality by defining their own sets and assigning them a GUID.

Referring now to Figures 7 and 8, an illustration of the process for connecting two kernel mode filters is illustrated. Figure 7 shows a logical block description wherein each filter instance and connection pin instance is represented by file objects. Figure 8 is a flow chart illustrating the steps to creating the file objects and the appropriate connections.

Beginning at step 144, an instance of Filter A 146 and an instance of Filter B 148 are created by a user mode agent. These are created using standard file system API for creating files with a particular device. Filter A 146 and Filter B 148 will be compliant filters or drivers because of their implementing the appropriate property, method, and event sets to support the creation of connection pin instances and for querying the respective filter's capabilities in terms of sets supported and connection pin factories defined for that filter.

The third party controlling agent will then query Filter A 146 and Filter B 148, respectively, at step 150 to determine connection pin factories available and the attributes for connection pin instances that may be created therefrom. These attributes include, as mentioned previously, the connection format and the data format for each individual type of pin instance for each respective filter 146 and 148. The querying will be accomplished using the set based query mechanisms explained in detail previously.

After querying such information, the third party controlling agent will determine the optimal connection format based on the ranges of data formats and connection formats previously queried. This determination occurs at step 152 and places in the third party agent the ability to use the same filters in different ways according to the needs of a selected connection path. The third party controlling agent will use the data intersection property,

topology information, and connection pin factories on both the filters in order to determine how best to select data format and connection arrangements depending on the actual filter graph being made.

Input filter pin instance 154 is created by the third party agent at step 156 using the optimal detection formation determined at step 152. Since input pin instance 154 is a file object, a handle will be returned from the create process that can be used for delivering I/O requests to the input instance 154. Furthermore, the creation of the input pin instance 154 was validated and uses the routing and validity mechanisms shown previously in discussion with Figures 4A-4C, 5, and 6.

In order to finalize the connection, output pin instance 158 is created at step 160 using as a parameter in the NtCreateFile call the handle of the previously created input pin instance 154. The effect of thus creating the output pin instance 158 is to utilize the system file management and I/O management facilities to create an internal IRP stack structure that allows an original write command to be consecutively processed by the variously connected connection pin instances and filters in an appropriate order so as to facilitate direct data flow between the differing filters. This requires that the input pin instance be created prior to the associated output pin instance that will be feeding the input pin instance.

The stack depth parameter of a device object controls how many stack locations are created from IRP sent to this driver. A stack depth parameter is assumed to be one when a device object is initially created and may be modified thereafter depending on the whether multiple drivers are chained together. In the current system, modification occurs, if necessary, when an output pin instance transitions from the initial "stop" state to the "acquire" or other state. Connection pin instance state transition is the mechanism that determines correct stack depth parameter information for proper IRP creation and treatment.

In order to correctly allocate the internal IRP stack structure for a chained set of connection pin instances, it is necessary to transition the connection pin instances out of the

stop state in a specified order; beginning with the last input pin instance (in this case input pin instance 154) and working consecutively backwards to an associated (*e.g.*, connected) output pin instance (in this case output pin instance 158). If many filters are chained together, the deepest filter's or bridge's input pin instance must be the beginning point of transitioning and building successively backwards until the initial output pin instance on a bridge or filter is set. In other words, the transition out of the stop state must occur backwards up the chain so that each connection pin instance gets the stack size needed after the previous connection pin instance. Typically, though not necessarily, a connection pin instance transitions from the stop state to the acquire state and for discussion purposes hereinafter transitioning to the acquire state will accomplish the same purpose with respect to stack depth parameter adjustment as transitioning out of the stop state.

Once all pin instances are in the acquire state, stream reads and writes may be issued to the filter graph. It is interesting to note that the system explained herein allows connection of associated input and output pin instances to occur in any order; only the transition from the stop state must occur in bottom up or deepest first fashion. Furthermore, the filter graph is reconfigurable to allow changes to be made after initial creation. When changes are made, state transitions need only occur on those connection pin instances that are in the stop state in order to assure correct stack depth parameter information.

Connection pin factories found on filters represent places where a filter can consume and/or produce data in a particular format. For example, a particular connection pin factory may support a number of different data formats, such as 16 bit 44 kilohertz PCM audio or 8 bit 22 kilohertz PCM audio. As explained previously, the connection pin factories and their different capabilities such as data format can be queried from the filter using the appropriate property set mechanism and the system I/O facilities. Actual connection pin instances are created based on the information received from the pin factories.

In a streaming environment, where a single stream write or stream read operation from a user mode agent will cause successive processing of the data through the connected filters, two main methods for IRP control can be used as part of the native facilities of the NT operating system. First, a separate IRP may be created by each filter and sent to the next filter for processing which will in turn create a new IRP for further processing down the chain. The other method is to use a single IRP and pass it between the successive filters using standard procedures provided for interacting with the I/O manager. If the first method of creating new IRPs for each successive filter in the chain is used, interconnection order between the filters is unimportant since the filter need only know the destination of the IRP in order to call the I/O manager and send the IRP to the designated filter. If an IRP is reused, it is important that the connection pin instance transitions from the stop state be made beginning from the last filter to receive the reused IRP for processing backwards up to the first filter to receive the reused IRP or to the filter that created the IRP for processing.

The current embodiment and implementation of the interconnected kernel mode filters utilizes IRP sharing advantageously to ease complexity in driver development, allow more robust drivers to be created, and provide more efficient processing. The "bottom up" pin instance state transition path will ensure that the proper stack order is created in the IRP processed by the successive drivers and that each driver object has the appropriate stack depth parameter set. Furthermore, the current state of the receiving input pin instance is checked in order to assure that the state transition sequence has been properly followed. For this reason, the communications property of a particular connection pin factory will determine the potential flow direction and aid in properly distributing the state transition of connection pin instances.

When creating an output pin instance (or IRP source), a reference to a file object representing an input pin instance (or IRP sink) on another filter will be passed as part of the NtCreateFile call. The appropriate create handler will be executed as explained previously

using the multiplexing dispatch function and device object/file object hierarchy. This create handler will have access to the device object of the filter having the input pin instance (*e.g.*, Filter B 148 in Figure 7) by way of the input connection pin instance file object (*e.g.*, input pin instance 154). From the device object, the previous stack depth parameter can be read, and the stack depth parameter of the device object for the filter having the output pin instance may be incremented. For example, the device object associated with Filter A 146 will have a stack depth parameter incremented from that of the device object associated with Filter B 148 for the connection illustrated in Figure 7. This normally occurs when transitioning out of the stop state and IRPs are not forwarded while a connection pin instance is in the stop state.

When a filter processes an IRP, it knows which stack frame or location within the IRP stack to access containing information designated for that particular filter by making reference to and using the stack depth parameter of the associated device object. Furthermore, the current filter will prepare the IRP for the next filter in the processing chain by decrementing the device object stack depth parameter to locate the next filters IRP stack location.

The filter code is responsible for preparing the next location in the IRP stack and for calling the I/O manager to pass the IRP to the next filter as designated. In this manner, the filter may designate which file object representing a particular connection pin instance is to receive the IRP and the associated data for processing. Hence, the standard I/O manager calls such as IoAttachDevice to stack the respective device objects for sequential processing of IRPs are not used.

It is noteworthy that creating a connection between connection pin instances does not imply creating new device objects to represent the connection. A single underlying device object is used to support an instance of a filter and all connection pin instances on that filter. Specific information necessary for proper data processing is kept within the context

area of the file object allowing the context information to be preserved while non-page memory use is kept at a minimum. It is also noteworthy that while an IRP-based medium has been illustrated, other mediums for communication between the interconnected filters may be used, such as direct function calls on non-host hardware-to-hardware communication.

Referring now to Figures 9 -10 and Figure 1 , the proper creation, connection, and state transition order of the software drivers as shown in Figure 1 (prior art) and Figure 2 (higher level logical diagram of the interconnected kernel mode drivers) are presented. Figure 9 illustrates the logical structure encompassed by box 162 and the processing steps contained therein. Figure 10 shows the creation of the connection pin instances to complete the interconnection of kernel mode filters and comprises the processing steps encompassed by box 164 on the flow chart shown in Figure 11.

When in the state of Figure 10 , having all interconnections made, the kernel mode filter system is ready for reads and writes in order to effectuate processing. The I/O system will use the IRP stack information properly set by the correct state transition process in order to pass the stream reads and writes onto the differing filter elements by way of their respective connection pin instances. It may be noted that some external software other than the agent used to create the graph, including a bridge or filter itself, as well as hardware will provide data for the stream reads and rights.

After beginning at step 168, the controlling agent 170 will create instances of reader filter 172, decompressor filter 174, effects filter 176, and sound rendering filter 178 at step 180. Furthermore, attachment will be made between reader filter 172 and a disk driver 182 in order to bring the data in from off of the disk drive. Creation of each filter instance is achieved by the user mode controlling agent 170 by using standard I/O calls to open a file on the appropriate device as found in the device I/O directory hierarchy. Such a call will return a handle to a file object representing the instance of each filter.

At step 184, the third party agent will query the effects filter 172, the decompressor filter 174, the effects filter 176, and the sound rendering filter 178 to determine the connection pin factory capabilities. These capabilities include what kinds of input and output pin instances may be created, how many instances of each connection pin factory the particular filter will support, the data format supported on each connection pin factory, the medium or type of communication path, etc. The capabilities are "queried" using the property set mechanism explained in more detail previously and the kernel mode filters are presumed to be compliant to the architecture since they support appropriate "sets" (*e.g.*, property set).

All such query information at step 184 will be used to determine if a chained connection path is possible between the respective filters by creating and connecting the appropriate connection pin instances. The third party agent will determine the types of pin instances needed for interconnection in order to make a filter graph to accomplish a given purpose.

The determination of the connection format based on the supported data formats is determined at step 186. Using the topology information, data format, and data intersection properties on the filter, a hypothetical filter graph may be created. Since connection order is not significant, this need not be done but could save time when trying to build a filter graph. Should this hypothetical filter graph be created without error, the third party agent will be assured that creating an interconnecting connection pin instances can be done with reliability. Because some queries will return errors unless an actual pin instance is created, it may be necessary to create such connection pin instances before a hypothetical filter graph can be created that will return a reliable indication of viability. Again, the hypothetical filter graph may be tested before any interconnections take place.

Once the correct connection information is known, as determined at step 186, the input pin instances may be created and interconnected and as represented by the loop of

processing steps enclosed by box 164 on Figure 10. This loop contains processing steps that will begin at the input pin instance furthest away from the source of the data stream. This last input pin instance is referred to as the "deepest" pin instance and may be created first, followed by the output pin instance associated therewith. A connection, therefore, is the creation of an output pin instance using the handle of a previously created input pin instance.

The pattern continues, with every input pin instance created successively afterwards prior to connection with the associated output pin instance. Such a connection scenario is given by way of example and is not to be limiting of other possible ways of connecting the respective output and input pin instances to form a connection between kernel mode filters according to the present system. The filters may be connected in any order according to implementation as long as the handle from the input pin instance is used during creation of the connected output pin instance on another filter. Furthermore, as explained previously, changes may be made to the filter graph after initial creation (and even use).

In the first iteration of the loop, input pin instance 188 will be created at step 190. After receiving the handle from the create function, the third party controlling agent 170 will use that handle as a parameter in an NtCreateFile call in order to create output pin instance 192 at step 194. By doing this through the first iteration, the sound rendering filter 178 is effectively connected to the effects filter 176 through the corresponding connection pin instances 188 and 192, respectively. In the current implementation, the NtCreateFile call is "wrapped" as part of a function call in an API made available to the user mode clients. This relieves the user mode developer of third party agents from needing to know as much detail and allows all relevant functionality be concentrated in a single user mode API.

At step 196, the third party agent determines if there is any other existing input pin instances to be created. If there are, an input pin instance must be created followed by the corresponding output pin instance on another filter. Eventually, all connections will be made

and the third party controlling agent 170 will prepare the filter graph for streamed data processing.

In this fashion, input pin instance 202 will be created on the second iteration of the loop enclosed in box 164 at step 190 while the output pin instance 204 will use the handle of input pin instance 202 as part of its creation at step 194. Finally, on the third and final iteration for this particular example, input pin instance 206 will be created followed by output pin instance 208 to finalize the connection.

At step 197, the third party controlling agent 170 will transition each connection pin instance from the stop state to the acquire state in preparation for streamed data processing through the filter graph. To correctly set the stack depth parameter in each of the device objects for the respective filters, it is necessary to make the state transition beginning with the "deepest" or last connection pin instance (*e.g.*, the last input pin instance to receive data for processing) and sequentially moving "up" the chain of interconnected kernel mode filters until arriving at the first connection pin instance (*e.g.*, the first output pin instance that will provide data into the graph). The first filter or bridge will create the IRP with enough stack locations allocated so that the IRP may be passed successively through each kernel mode filter in the graph in an efficient manner.

Finally, the third part controlling agent 170 issues the stream reads and writes in order to process the data at step 198 before ending at step 200.

As explained previously, each creation of an output pin instance will require the handle of a file object representing the input pin instance to be connected thereto. This file object reference will allow the create handler for the output pin instance to save a reference to the device object corresponding to the input pin instance for current or future access.

More particularly, this allows the stack depth parameter of the device object managing the input pin instance to be accessed by the driver of the output pin instance during state transition from the stop state to the acquire or other state. The value of the stack depth

parameter associated with the input pin instance is accessed, incremented, and saved into the stack depth parameter for the device object corresponding to the output pin instance.

The stack depth parameter is used to determine where in the shared IRP stack structure the stack frame information is located for a particular filter and will be different for each filter. By so interconnecting the filters and making the state transition in proper sequence, a single IRP may be passed down the chain of interconnected filters in kernel mode with no necessary communication into user mode.

It may be noted that it is possible to have multiple instances based on the same connection pin factory. For example, an audio mixing filter may mix multiple input pin instances into a single output pin instance in terms of processing. Each input instance is of the same type and the filter may only support one type of input pin. Such an arrangement would also be an example of having multiple inputs to a single output.

The converse is also true wherein a splitter filter may have a single input connection pin instance while providing multiple output pin instances thereby multiplying streams of data. Those skilled in the art will note that many variations and useful combinations can be made from the connection mechanism explained herein according to actual implementation and the needs thereof.

The uniformity and standardization achieved by requiring all compliant filters to support a common mechanism (*e.g.*, property sets, methods sets, and event sets) that can be independently implemented by driver developers allows a controlling agent to conveniently connect compliant filters provided by various different software providers. Furthermore, many of the facilities in terms of connection pin factories needed in one circumstance may not be needed in another circumstance. A determination of the necessary connection pin instances is made initially by the third party controlling agent that makes the actual interconnections between different filters.

Referring now to Figure 11A, operation of a buffer allocator mechanism is shown as used between multiple processing components. Also shown is the data flow *(i.e.* actual data transfers between buffer frames) between buffers associated with particular processing components. While control will pass to each processing component, data will be transferred only when necessary with some processing components performing their data manipulations and returning the data to the existing buffer frame. In other words, data may be processed in the same location without being transferred to a new buffer and is said to be processed "in place."

A sink processing component 210 will have a buffer allocator mechanism 212 (represented by a square) as part of its functionality. A buffer allocator mechanism will only be present in processing components where it is necessary to ensure certain placement of the data into specific memory such as on board memory on a sound or video processing card or where the previous buffer has unacceptable characteristics, such as byte alignment, frame size, etc. Furthermore, references to the buffer allocator mechanism to be used for allocating frames of buffer memory is indicated by a circle and all processing components will have such references. Note that the source processing component 214 has a buffer allocator reference 216 that references the sink buffer allocator 212 as represented by arrow 218. Furthermore, the transfer processing component 220 has a null buffer allocator reference 222 and the sink processing component 210 also has a null buffer allocator reference as indicated by the empty circle 223.

In this simple processing example, the source processing component 214 allocates a buffer frame 224a by using the sink buffer allocator 212 and accessed using buffer allocator reference 216. The allocated frame 224a is filled with data by the source processing component 214 as represented by arrow 226. It may be noted that the source processing component may perform some data manipulation or transformation before writing the data into the newly allocated frame 224a.

At this point, the source processing component 214 has finished processing and turns control of processing to the transform processing component 220 as represented by arrow 228. The transform processing component 220 will do no buffer allocation or transferring of data from one buffer to another since no buffer allocator mechanism is indicated since buffer allocator reference 222 reference has a null value. Therefore, the transform processing component 220 performs an "in place" data transformation in the allocated buffer frame 224b as represented by arrow 230.

Since the data has not been transferred to a new buffer frame, buffer frame 224a, frame 224b, and frame 224c are the same frame and are simply passed in succession to different processing components. Arrow 231 represents passing the allocated frame between source processing component 214 and the transform processing component 220.

Finally, the transform processing component transfers control of processing to the sink processing component 210 as represented by arrow 232. It may be noted that along with processing control, the same frame is passed for processing as shown by arrow 234 between the frame 224b and 224c. Again, as depicted herein, frame 224a, frame 224b, and frame 224c are all the same frame allocated originally by source processing component 214 and are shown separately for illustration purposes.

The sink processing component 210 will finish processing the data and free the allocated frame 224c within a buffer as presented by arrow 236. Since the sink component 210 is no longer using the buffer, the arrow 236 points inward to the sink processing component 210 and the frame may now be deallocated or reused.

Figure 13 shows how a buffer allocator mechanism would be logically implemented in the scheme of interconnected kernel mode buffers that has been explained previously. Figures 12 and 13 both represent the same filter graph topology and are used to better illustrate operation of the buffer allocator mechanism. The relevant drivers and portions thereof each have access points that allow user mode clients to control the driver and

are represented by file objects. Intercommunication occurs using IRPs (whether created by kernel mode drivers or by the NT executive in response to user mode I/O operations).

An instance of source processing component 238 (represented by a file object) has associated therewith an output pin instance 240 (also represented by a file object) that provides a source of connection to another filter instance. An input pin instance 242 that is a "child" of a transform filter 244 was created having a reference to the output pin instance 240 in the manner explained previously in more detail. In like manner, a sink filter 246 having an input pin instance 248 is connected to an output pin instance 250, that is related to the transform processing component 244.

In the system of interconnected kernel mode software drivers, a buffer allocation mechanism is related to input pin instances and is said to be created or formed on the input pin instance. Furthermore, output pin instances will logically have reference to a buffer allocation mechanism, if necessary, and the filter having the output pin instance will utilize that reference to make any buffer frame allocations and data transfer to new frames prior to turning control to another processing component. As explained, a null reference will indicate that no data transfer to a new frame is necessary and that processing may occur in the existing frame, (*i.e.*, after processing, the data is returned to the same buffer frame). Whether a buffer allocator reference is present is determined by the initial negotiation of the third party controlling agent that created the filter graph.

The buffer allocator mechanism formed on an input pin instance 248 is represented by a file object while the dashed line 254 shows that the output pin instance 240 has reference (*e.g.*, a pointer or handle) to the file object representing the buffer allocator 252. In the example shown in Figure 13 , frames of memory are allocated from system memory 256.

Since the filters may be interconnected in a number of different ways, a buffer allocator though available, may not be necessary. A file object representing an instance of

a buffer allocator will only be created should the third-party controlling agent interconnecting the filters determine that it is necessary. In this manner, filters may flexibly be connected in a variety of configurations and still maintain optimal data transfer characteristics. Furthermore, default system buffer allocators can be provided to further reduce the development effort of driver developers.

A third-party controlling agent will also query buffer requirements of connection pins as part of creating a hypothetical model before making actual filter connections. While some implementations may allow queries before pin instantiation, the present embodiment requires that the actual connection pin instance be created before the buffering requirements can be ascertained. Furthermore, the exemplary embodiment disclosed throughout is queried through use of the set mechanisms explained previously.

When a third party client or controlling agent completes the interconnections of kernel mode filters to make a filter graph, it then initiates negotiation of allocator requirements using the handles of the input pin instances (or sink pin instances). By convention, the input pin instances define buffer allocation requirements and provide a buffer allocation mechanism while the output pin instances have reference to any appropriate buffer allocation mechanisms associated with the input pin instances. Those skilled in the art will understand that other conventions may be used to effectively accomplish the same results including reversing the buffer allocation responsibilities between the input and output pin instances.

Buffer allocation requirements are ascertained by using a particular property set mechanism that will be supported by all compliant filters. It may be noted that the "buffer allocator" property set may be organized in a number of different variations as may any other set. For example, the property set may have a single property wherein the property is a data structure having segmented information or the property set may have multiple properties, one for each distinct framing requirement element. A single property composed of a data

structure is more efficient in some circumstances since fewer property set queries are necessary by the third party controlling agent in order to retrieve all the framing requirement information. Furthermore, a single data structure could be used not only to query requirement information but also for specifying parameters when an actual buffer allocator is created.

The table below shows a non-exhaustive list of framing requirement elements that may be included in a data structure or as individual properties. Also included is an explanation of how such an element would be used in an exemplary embodiment.

TABLE 3

Buffer Allocator Framing Requirement Elements	
Element	**Description**
Control Options (Create)	This element contains control options that are specified during buffer allocator creation on a given connection pin instance. Some options include: *System Memory:* This control option indicates the use of system memory for buffer frame allocations. When specified, the buffer allocator must allocate memory from the pool (as specified in the allocation pool type element) located in system memory. If this control option is not specified, it is assumed that the input connection pin instance (or sink pin) provides a system address mapping to on-board RAM or other form of storage on a device attached to the system. Such a device may be an add in card, a PCMCIA card, etc. *Downstream Compatibility:* This control option specifies that the allocator framing elements of the buffer allocator being created are compatible with the downstream allocator. This option is normally specified when an in place modifier is assigned an allocator for copy buffers. If the filter is not required to modify a given frame, it may submit the frame to the downstream filter without allocating an additional frame from the downstream allocator .

Requirements (Query)	This element contains the allocator requirements for a connection pin instance (input or sink) that are returned during the query of a connection point. Some potential requirements include: *In-place:* This indicates that a connection pin instance has indicated that the filter may perform an in-place modification. Otherwise, a buffer must be allocated for receiving the modified data. *System Memory*: The connection pin instance (input or sink) requires system memory for all buffer frame allocations. If this requirement is not set, it is then assumed that the connection pin instance provides a system address mapping to on-board RAM or other form of storage on a physical device. *Frame Integrity:* This requirement indicates that the connection pin instance requires that downstream filters maintain the data integrity of specified frames. This may be used when statistics are required of a previous frame in order to properly process the current frame. *Allocation Required:* This requirement mandates that connection point must allocate any frames sent. *Preference Only Flag:* This requirements flag indicates that the other requirements signalled are preferences only. This means that the requirements indicate a preferred way of operation but the connection point will still operate correctly and process frames which do not meet the specified requirements.
Allocation Pool Type	This element determines the allocation pool type from which kernel mode entities will receive system memory for buffer frame allocation.
Outstanding Frames	This element indicates the total number of allowable outstanding buffer frames that can be allocated. In this manner, the total size of the buffer can be controlled. Furthermore, specifying a zero for this element indicates that the filter (through a particular connection pin instance) has no requirement for limiting the number of outstanding frames.
Frame Size	This element specifies the total size of a buffer frame. Again, specifying a zero for this element indicates that the filter (through a particular connection pin instance) has no requirements for this member.
File Alignment	This element specifies the alignment specification for allocating frames on certain boundaries. For example, an octal byte alignment may be specified. Furthermore, in the exemplary embodiment, the minimum alignment capability is on quad boundaries providing optimal data access speed on current PC hardware architectures.

Those skilled in the art will appreciate that there may be other relevant framing properties that may be included. Furthermore, the buffer allocation mechanism as described herein functions in substantially the same manner regardless of whether more buffer frame elements than specified in Table 3 are included or whether a subset of those specified are implemented.

When the filter graph requirements have been determined by the third party controlling agent, then the appropriate kernel mode buffer allocators may be created on the appropriate input connection pin instances. The client creates a buffer allocator instance using the handle of the appropriate connection pin instance and by specifying the appropriate creation parameters for the buffer allocator. In this manner, the resulting file object that represents a buffer allocator will be a child of the connection pin instance and will use the context information from that file object and the file object representing the instance of the filter itself for its proper creation.

In other words, the same mechanism explained previously for validating creation of a connection pin instance and for routing messages to the appropriate handlers for a given connection pin instance will apply in like manner to the creation of an instance of a buffer allocator. Again, the NtCreateFile call will be wrapped in a higher level function call as part of an API that third party controlling agent may use.

Buffer allocator instances will be created, if at all, only on input pin instances in the presently explained embodiment.

If a buffer allocator instance handle is not specified to an output connection pin instance handle through an appropriate API call, then the filter can assume that the stream segment submitted via stream I/O controls meet the filter's requirements and it may freely modify the data in-place.

A system supplied default allocator may be used by filter developers to simplify the task of providing buffer allocation capabilities to input connection pin instances. The default allocator provides for a system memory buffer frame allocation for those device drivers that are able to transfer data from system memory but require specific memory allocation properties. By using the default buffer allocator, a filter developer is relieved from the task of actually providing code to do the buffer allocation. The filter will still be written, however, to support the buffer allocation requirements request by supporting an appropriate property set.

To use the default allocator, the filter designer will (1) provide code to respond to the buffer allocation requirements requests, and (2) place a default allocator creation handler reference in the validation table of the particular connection pin instance to which the default allocator pertains. In other words, when a create request for the allocator type mechanism is submitted through the filter, a specific GUID string is matched in the validation table that in turn allows access to the default allocator creation handler.

The default buffer allocator uses system memory and will operate in accordance with the buffer allocator framing properties submitted as part of the creation request. Such a default allocator is likely to be used by various transform filters depending upon their particular function and interconnection order.

Filters requiring buffer allocators for on-board memory or other device dependent storage methods can provide a specific buffer allocator by supporting a buffer allocator property set and method set. For a filter specific allocator, the filter will be responsible for providing program code to implement the entire functionality. The operation of the buffer allocator, however, will be the same as for every buffer allocator, whether default or filter specific, so third party agents may interconnect the filters and buffer allocators properly.

The file object representing the buffer allocator, when successfully created, has contained in the file context area a pointer to a data structure. This data structure will contain

a dispatch table for directing IRPs to designated handlers based on the various IRP codes (*e.g.* create, I/O control, etc.) as well as other data areas and structures for maintaining the state of the buffer allocator. Some implementation dependent information that may be tracked includes reference to the file object of the connection pin instance to which the buffer allocator pertains, reference to an allocation framing requirements data structure, an event queue of those clients waiting for events, a queue of those allocation requests (received by an IRP, for example) that are outstanding, etc.

There are two interfaces, or ways of communicating, available to the buffer allocation mechanism disclosed herein in the exemplary embodiment. First, all allocators must provide the IRP-based interface in order to communicate properly with user mode clients. Optionally, if the allocation pool type can be serviced at the dispatch level (a raised priority level at which a limited subset of services are available, but at which lower priority tasks are blocked out on that processor) of the operating system, a function table interface may be supported allowing interconnected filters to use direct function calls (during DPC processing) for higher performance. This allows event notifications to be communicated between the interconnected filters without the extra overhead of passing an IRP through the I/O manager, or scheduling a thread of execution and waiting for a context switch.

The IRP-based interface will successively process IRPs in the following fashion. When an allocate request is submitted, the buffer allocator will complete the request and return a pointer to an allocated buffer frame or if all frames have been allocated, the allocator marks the IRP pending and adds the IRP to the allocator's request queue for processing, in approximately FIFO order. Finally, the buffer allocator will return status pending to the caller.

When a buffer frame is made available to the allocator, the allocator attempts to complete the first request in the request queue the for IRP-based interface. Those IRPs that could not be serviced previously will be in this request queue and will be waiting for the

newly freed frame for processing. Otherwise, if no work awaits in the request queue, the frame is returned to the free list.

The dispatch level interface using the direct function call table operates in the following fashion. When an allocate request is submitted by making a function call (which may be done during a DPC), the allocator returns a pointer to the frame if one is available or returns null immediately. The kernel mode requester may then wait for a free event notification to know that there is a free frame available. Upon receipt of this notification, the kernel mode requester will attempt the allocate request again.

Note that it is possible for both the dispatch level interface and the IRP-based interface to contend for an available free frame. Also, if there are allocation request IRPs waiting to be completed in the request queue, the allocator must schedule a worker item if the current IRQL is not at the passive level when the caller frees a frame since using the direct call interface implies the possibility of being at the DPC level. Essentially, the waiting queue of IRPs does not find out about the free frame until the worker item has been run.

Furthermore, the buffer allocation mechanism explained herein is adaptable to be used with MDLs (memory descriptor lists) in order to further reduce inter buffer transfers. Such an implementation would allow greater seamless interaction with the system memory facilities of the NT operating system.

Referring now to Figure 14, the interconnected filter system of Figures 9 and 10 are shown utilizing the buffer allocation mechanism disclosed previously. Once the controlling agent 170 has made interconnections between the filters, it will query each input pin instance for its buffer requirements. As shown herein, the sound rendering filter 178 will need to allocate buffer frames from the sound card memory 258 and the decompressor filter 174 will allocate memory from system memory 260 which may receive the data directly from disk drive 262.

Controlling agent 170 will create a buffer allocator 264 that is represented by a file object and being formed on the input pin instance 188. The buffer allocator 264 will allocate buffer frames from the sound card memory 258 and a reference to the buffer allocator 264 will be set in the output pin instance 204 as represented by dashed line 266. This reference will be the handle to the file object representing buffer allocator 264 and will be used by the decompressor filter 174 to allocate buffer frames as necessary before transferring data to the new buffers.

In like manner, buffer allocator 268 is also represented by a file object and created or formed on input pin instance 206. This buffer allocator 268 will manage allocation of system memory 260. The controlling agent 170, after creating buffer allocator 268, will store a reference thereof in output pin instance 208 as represented by dashed line 270. Again, buffer allocator 268 will be responsible for allocating buffer frames between system memory 260 so that data may be transferred therein from disk 262.

The controlling agent will also put a value of null for the buffer allocator reference of output pin instance 192 to thereby indicate that an in place transformation will occur wherein the effects filter 176 will read the data from the existing buffer in sound card memory 258 and replace the data back into the sound card memory 258 after applying whatever transform or effect is required. Alternatively, if the controlling agent does not set the buffer allocator reference, it will be assumed to have a value of null and an in place transformation will occur.

Referring now to the flow chart of Figure 15 and the logical diagram of Figure 14, operation of the buffer allocators will be shown. This process occurs after the interconnections are made and the stream reads and writes are given from the controlling agent 170 to the reader filter 172.

Initially, the reader filter 172 allocates a frame in the system memory using the decompressor filter 174 buffer allocator 268 at step 272. The output pin instance 208 of

reader filter 172 will have a handle to the file object representing the buffer allocator 268 as represented by line 270, and will therefore have direct access to controlling the buffer allocator 268.

Once the file reader filter 172 has access to an actual buffer frame in system memory 260, it will fill the frame with data from disk 262 as represented by arrow 274 at step 276. It may be noted that the reader filter 172 may make a transformation or other manipulation on the data as it brings it into system memory 260.

The file reader filter 172 then initiates a stream write to the decompressor filter 174 at step 278. This stream write will be passed by way of an IRP through the NT I/O manager. At step 280, the decompressor filter 174 will allocate a frame of sound card memory 258 using the buffer allocator 264. The decompressor filter 174 will have knowledge of the buffer allocator 264 since a handle thereto was stored with respect to output pin instance 204.

The decompressor filter 174 will decompress the data and transfer the data to the frame previously allocated in sound card memory 258 as represented by arrow 284. Note that in decompressing the data there may be more frames allocated from sound card memory than exist in system memory. The extra ratio of buffer frames is necessary to accommodate the decompression effect on the data.

It is important to note that when transferring data from one buffer to another buffer that there may not be a 1:1 correspondence in terms of the amount of data transferred. In other words, the receiving buffer may require more or less space (or frames) depending on the nature of the filtering or transformation that takes place between the buffer transfer.

Once the decompressor filter 174 is finished decompressing a particular frame, it passes the stream write to the effects filter 176 using the facilities of the NT I/O manager. The effects filter 176 receives the stream write IRP at step 288 and processes the data in place in the existing sound card memory 258. Such effects processing would generally correspond to a 1:1 replacement of data requiring neither more or less buffer memory.

Once the effects filter 176 is finished processing the data in place, the stream write IRP is passed to the sound rendering filter at step 290. Again, the mechanism causing the stream write IRP to be transferred to the sound rendering filter 178 is the NT I/O manager facilities called by the effects filter 176.

Finally, the sound rendering filter 178 will receive the stream write IRP at step 292 and control the sound card hardware in order to form the actual rendering of the sound data existing in sound card memory 258. At that point, the sound card buffer frames that had been previously allocated can be reused to service writing requests or freed if there are no outstanding requests. The availability of a buffer frame is made known to the decompressor filter 174 so that any waiting stream writes may be used to process and place data into the freely allocated buffer frames. In like manner, the buffer frames of system memory 260 are either reused or freed.

Referring now to Figure 16, an overview system is shown wherein a master clock mechanism can be used to synchronize two data streams that are generated in real time and said to be "live." An audio renderer 296 receives a live audio stream 298 for processing as represented by arrow 300. As the audio renderer 296 processes the data, it is perceived as sound through a speaker 302 once the audio renderer 296 has converted the data samples into the appropriate electronic signals and sends the signals to the speaker 302 as represented by arrow 304.

In like manner, a video renderer 306 receives a live video stream 308 as represented by arrow 310 for processing. As a result of being processed by the video renderer 306, the data samples are converted into appropriate video signals and are transmitted to a monitor 312 as represented by arrow 314 where they may be perceived as images.

Typically, the live video stream 308 and the live audio stream 298 will come simultaneously into the rendering components. Since it is important to match the video with the audio when rendered at the speaker 302 and the monitor 312 in order to produce a

coherent presentation, processing of the live video stream 308 and the live audio stream 298 must be synchronized.

In order to accomplish this, one data stream is chosen to be the master. For purposes of this disclosure and explanation, the live audio stream 298 has been arbitrarily chosen to be the master data stream. Note that nothing prevents the live video stream 308 from being chosen as the master or any other stream of data for that matter. Furthermore, an external clock, based on a data stream entirely external to the filter graph, may be used to synchronize both the live audio stream 298 and the live video stream 308. Therefore, the master clock mechanism 316 is shown appended with a solid line to the audio renderer 296 and has a dashed line 318 to the video renderer indicating that the video renderer will in some manner receive synchronization information from the master clock 316 that is using the audio stream as the master.

The master clock 316 may also be used by the audio renderer 316 in order to match the rate of processing at the audio renderer 296 with the rate of the original audio stream 298. In other words, the live audio stream 298 was produced by another processing component, such as digitizing filter/digitizing hardware on another system. Since the audio stream is composed of digitized samples, the samples were produced at a rate according to the hardware and oscillator of an original processing component. This origination rate for the original processing component to produce the live audio stream 298 is found in or derived from the time interval information of the live audio stream 298.

Sample information in a data stream may contain time interval information in a variety of ways. One way is for each sample or group of samples to have a timestamp associated therewith. Furthermore, the format of the data itself may imply the sample rate. For example, by the definition of the data format, each sample may be taken at a certain time interval (*e.g.*, 1/30th of a second for a video sample or 1/22,050th of a second for audio data). Additionally, a timestamp may indicate a presentation time that is matched to a clock at the

rendering processing unit to determine when a particular sample is to be rendered or "presented." Also, a timestamp may indicate some other time value with the relative difference between time stamps giving time interval information. Finally, mixtures of data format and timestamp information may be used in order to provide the time interval information necessary for creating a time clock based on the position within a reference data stream being processed. .

It may be noted that "discontinuities" may be coded into the data stream that may be based on time or data gaps. A time discontinuity is where the stream simply jumps to another presentation time, such as when recording ended at a certain time and suddenly began at a later time. A data discontinuity is where the stream may have periods of repeated data that can be more efficiently coded in the stream, such as representing periods of silence by a given code followed by duration of the silence.

One time value provided by the master clock 316 is based on the time interval information found in a stream of data samples and known as media time. Media time is positional since it represents a position within the master stream, which in this case has been chosen as the live audio stream 298. Furthermore, when processing quits at the audio renderer 296, the media time is also frozen at the master clock 316. A discontinuity in the data stream will be processed to that "position" within the stream continues to advance while in the discontinuity by using the underlying hardware oscillator or other source to track the advancement of time. In this manner, multiple data streams may still be synchronized to a data stream having discontinuities of time or data therein.

By synchronizing the video renderer 306 to the media time found on the master clock 316, which in turn is based on the time interval information in the live audio stream 298, no overhead is necessary to manage stopped or paused states since media time advances ("ticks") only when the live audio stream 298 is being processed at the audio renderer 296.

It should be noted that the video stream 308 may also need to be rate matched as well as synchronized. The rate matching of the video stream 308 would occur in much the same manner as will be explained hereafter in connection with the live audio stream 298 and the audio renderer 296. Naturally, the live video stream 308 rate matching is based on the hardware oscillator (physical time) of the video renderer 296.

Also available at or otherwise provided by the master clock 316 is a physical time value that is based on the underlying hardware in the audio renderer 296. Physical time usually progresses regardless of whether the audio renderer 296 is actively processing audio data or is in a stopped or paused state. Furthermore, by sampling the rate of progression of the media time (representing the originating hardware processing rate) and the physical time (representing the actual processing rate at the audio renderer 296) adjustments may be made at the audio renderer to process the live audio data 298 so that the rates are relatively matched.

While rate matching is not an important issue when the rate of data production is under control of the processor such as reading stream data from a local disk or other storage device, it is crucial when processing a "live" data stream that continuously produces data and the production rate is out of the control of the processor. As used herein, a "live" data or media stream refers to a stream produced at a rate that is not controlled by the processing components and, therefore, must be processed completely in real time or lost. Examples of a live data stream include but are not limited to the following: a real time feed where the stream is produced in real time either locally or over a network connection, receiving the stream from a network connection regardless of whether it is a real time feed or previously stored data sent over the network, etc. In all cases, the stream must be processed as it is received.

If the live media stream produces too much data for the audio renderer 296 to process, the audio renderer 296 will eventually lose data due to finite buffering capabilities

in a condition that is known as data "flooding." On the other hand, if the live audio stream 298 produces data at a lower rate than the audio renderer 296, the audio renderer will suffer data "starvation" in that it will not have data to process. In either case, it is beneficial for quality rendering of the live audio stream 298 to make adjustments to match the rendering rate at the audio renderer 296 with the origination rate as represented in the time interval information in the live audio stream 298.

One way of making rate correcting adjustments is to periodically duplicate data samples so that the audio renderer will not become starved or to throw away samples so that the audio renderer will catch up with the live audio stream 298 that outproduces its capabilities. In either case, duplication or omission of data samples would judiciously occur at intervals over stream processing so that a minimal amount of impact is perceived at the speaker 302.

Rather than synchronize directly to the master clock 316, it may be useful to translate another clock's time base to match the media time found in master clock 316 or to translate presentation time information in another media stream to match the media time of the master clock 316. In order to facilitate such translation, the master clock 316 will provide a correlated media time value that consists of two separate time values, the media time value and a reference time value common to other components, and does so in an atomic (*e.g.*, unitary) operation to reduce the introduction of timing errors. A correlated physical time value is also provided.

The reference time is typically based on the PC clock which should be available to all processing components or other clock mechanisms found on the system though another clock could be chosen. The atomic operation may make use of native system facilities for implementation so that both values may be retrieved with the least amount of interruption or other interference. The correlated time (either media or physical) can be used by another processing component, such as a filter, to determine a relative offset or delta between the

media time and the reference time in order to make any necessary translation with a minimum of error introduced as will be explained in more detail hereafter.

Referring now to Figures 17 and 18, a clock mechanism as described above is implemented with the interconnected filter system explained previously. Figure 17 represents a live video stream processing set of filters that is synchronized to an audio stream and receives synchronization information from a master clock mechanism. Figure 18 is a set of interconnected filters for rendering a live audio stream that is used to generate the master time reference for a master clock mechanism.

Referring first to the video stream processing components in Figure 17, a live video stream 320 is received by a video reader filter 322 as represented by arrow 324. The source of the live video stream 320 may be from on-board digitizing hardware at the computer, packets of digitized samples received over a computer network, etc. The live video stream 320 is comprised of digitized samples that may be rendered into actual video images and the samples have time interval information associated therewith as explained previously. Again, according to the interconnected filter system explained herein, the video reader filter 322 is represented by a file object.

The video reader filter 322 has an output pin instance 326 that is connected to an input pin instance 328 on a video decompressor filter 330. Video decompressor filter 330 will receive the live video data in compressed format and decompress it into a format acceptable to video rendering hardware. After processing frames of video data, the decompressor filter will turn control over to the video rendering filter 332. A connection is made between the video renderer filter 332 and the video decompressor filter 330 by means of the output pin instance 334 and the input pin instance 336, respectively.

The input pin instance 336 will receive timing indications for synchronization from the master clock mechanism in the audio stream processing components (*see* Figure 18) as represented by box 338 and dashed line 340. This synchronization timing information

may be received in a variety of different ways. First, a controlling agent 342 that interconnects the filter graph topology shown in Figures 15A and 15B may have a clock mechanism 344 make event notifications based on either a specific position (*i.e.*, time) within the master audio stream or a specific time interval may be chosen for interval event notifications. Alternatively, the video rendering filter 332 may query the clock mechanism 344 (*see* Figure 1 8). Because the clock mechanism 344 is represented by a file object, the controlling agent 342 can make appropriate references to the clock mechanism 344 to the input pin instance 336 or the video renderer filter 332, thereby allowing the video renderer filter 332 to query the clock mechanism for correlated time values in order to make translations, if necessary, or media time values to assure that processing is synchronized.

Finally, the video renderer filter 332 controls the video hardware 346 and transfers data thereto as represented by arrow 348. The video hardware will render the data and generate video signals causing images to be displayed on monitor 350.

Referring now to Figure 18 , the processing components of the live audio stream are now explained in detail. Again, the same controlling agent 342 will create the respective filter instances along with any necessary input pin instances and output pin instances in order to make the interconnections between the respective filters and it is the controlling agent that makes the entire kernel mode filter graph topology.

The live audio stream 352 is initially brought into the filter graph topology by the audio reader filter 354 as represented by arrow 356. The data is decompressed by the audio decompressor filter 358 that is connected to the audio reader filter using respective output pin instance 360 and input pin instance 362. Again, each filter and connection pin instance described herein is represented by a file object and created as explained previously.

The audio decompressor filter is connected to the audio renderer filter 364 through the output pin instance 366 and the input pin instance 368, respectively. Also associated with the input pin instance 368 is an instance of a clock mechanism 344 that will provide or

send indications of media time, physical time, and/or correlated times to other entities, particularly the input pin instance 336 for the video renderer filter 332 (*see* Figure 17) as represented by box 370 and dashed line 372. The actual implementation and availability of the clock mechanism 344 will be explained in more detail hereafter. By using the clock mechanism 344, other data streams may synchronize processing to the processing of the "master" or "reference" audio stream 352 as illustrated in Figure 18 .

The audio renderer filter 364 controls the audio hardware 374 and will pass control information and data as represented by arrow 376. The audio hardware, under the control of the audio renderer filter 364, will render the digitized data samples into real time electronic signals that may be transmitted to and perceived from a speaker 378.

When creating interconnected kernel mode filters as explained previously, each filter instance is created from a "device" available to the system. In this manner, the video reader filter 322 and the audio reader filter 354 may be separate instances of the same file reader device.

Additionally, because a particular filter may support many connection pin factories for creating connection pin instances, the video decompressor filter 330 and the audio decompressor may be instances of the same decompressor device. When the controlling agent creates connection pin instances on the respective filter instances, different pin factories are chosen as templates for creating the connection pin instances depending on the type of data being processed through the filter. This is the inherent flexibility in the interconnectable kernel mode filter system disclosed herein.

In order to form clock mechanism 344, a controlling agent 342 will typically query a connection pin instance, such as input pin instance 368 on the audio renderer filter 364, to determine if a clock mechanism is available. Other implementations may require the controlling agent 342 to simply attempt creation of a clock mechanism on a connection pin instance, thereby determining clock mechanism availability by trial and error. Furthermore,

a clock mechanism may exist in user mode and send notifications, etc., to input pin instances of filters in kernel mode through a kernel mode proxy of that clock or some other mechanism.

By chosen convention in the exemplary embodiment, timing event notifications and clock mechanisms are typically found in association with input connection pin instances. Clearly, however, any entity represented as a file object may receive event requests by way of IRPs or by a direct procedure call after retrieving the direct procedure call table through a previous IRP request.

A system supplied default implementation of a clock mechanism may be supplied or made available to software filter developers in order to provide timing functionality without requiring additional code development. Formation of any clock mechanism (supplied default or custom) occurs by specifying a GUID string value in the create request that will be used in a connection pin instance validation table in order to access the correct creation handler for forming the clock mechanism on the connection pin instance. The process of file object creation, validation, and designation of IRPs was discussed in more detail previously in connection with Figures 3, 4A-4C, 5, and 6. In the same fashion as creating a connection pin instance, a clock mechanism create request may be properly routed and validated.

In order to use the default clock supplied by the system, a driver developer will call the default clock create method from within the create handler that they create and provide to the system. The create handler is accessed from the context area of a file object representing a connection pin instance in the manner described previously. The driver developer, therefore, implements a driver specific clock mechanism, and the appropriate driver specific create handler would be placed in the validation table. Complexity of create handler development is reduced by making a call to the default clock create method rather than coding and implementing the clock functionality.

For a clock mechanism to be made according to the interconnected filter system shown throughout, it must support a certain property set and event set. Naturally, the default clock mechanism supports the specified sets and any filter specific clock mechanism would require the filter developer to write and provide code to implement the clock mechanism property set and event set. In this manner, any controlling agent having a handle to a clock mechanism may query for support of the specified set and know how to operate the clock mechanism. Table 4, below, indicates a number of properties that may be included in a clock mechanism property set.

TABLE 4

Clock Mechanism Properties	
Properties	**Description**
Media Time	This property returns the current media time on the clock as represented in standard units. For example, in the exemplary embodiment, this would be timed in 100 nanosecond units. In the common case where the clock mechanism is being presented by some processing component, such as a filter or connection pin instance on a filter, which is processing a timestamped data or control stream, the media time value returned by this property reflects the current position in the stream. Furthermore, any data or control stream having time interval information may be used to allow the clock to properly operate. If the underlying processing component becomes data starved, the media time as presented by this property also stalls. Furthermore, if the data stream or control stream is comprised of timestamped data, and the timestamps on the stream are modified, the media time as presented by this property will also reflect this modification. Therefore, if timestamp information is "translated" from one time frame to another, any clock mechanisms based on that data stream will also be translated. Finally, if the underlying component is directed to stop or pause processing the data or control stream, the media time as reflected by this property will also stop.
Physical Time	This property returns the current physical time for the clock mechanism. The physical time is a continuously running time which is based on some underlying hardware oscillator. For the default clock mechanism, this would be the PC clock. For a filter specific clock mechanism, it may be the actual rendering hardware clock. Physical time does not stall when stream starvation occurs and may not stop when stream processing is stopped or paused by the underlying processing component. The physical time rate of progression matches that of the underlying hardware and therefore allows an interested party to compare the rate of the physical time with the rate of its own processing time or media time progression in order to match processing rates. In concert with querying the current media time, and with feedback as to stream processing starvation or flooding, a processing component can not only make adjustments for matching processing rates, but also determine if it is ahead or behind a desired stream position for the master or reference stream (*i.e.*, synchronize).

Correlated Media Time	This property returns both the current media time value for the clock mechanism, and a corresponding reference time value as a single atomic operation. In the exemplary embodiment, the PC time serves as a reference time that is universally available to many different operational entities. By using the correlated media time, a processing component using a different clock may perform a translation using the common PC time while introducing a very small, and in many instances insignificant, amount of error. Since translations back and forth would use the same atomic operation, the small error is not cumulative.
Correlated Physical Time	This property returns both the current physical time value for the clock mechanism, and a corresponding reference time value as a single atomic operation. In the exemplary embodiment, the PC time serves as a reference time that is universally available to many different operational entities. By using the correlated physical time, a processing component using a different clock may perform a translation using the common PC time while introducing a very small, and in many instances insignificant, amount of error. Since translations back and forth would use the same atomic operation, the small error is not cumulative.
Granularity & Error	This property returns the clock increment granularity or resolution in order to indicate how precise or fine each clock "tick" will appear. In the exemplary embodiment, the granularity value is the number of 100 nanosecond increments per each clock tick. If a clock runs at or under 100 nanosecond resolution or granularity, the value would simply return 1. The granularity property allows a client of the clock to react differently depending on the resolution. The property further provides an error indication in 100 nanosecond increments with 0 representing the least amount of error.
Parent	This property returns the unique identifier of the processing component which produced the clock mechanism. This allows a client or processing component to determine if the clock mechanism is its own clock mechanism or some other processing component's clock mechanism so that it may react differently as the circumstances dictate. For the exemplary embodiment, this property would be a handle to the file object of an input connection pin instance or filter upon which the clock mechanism was produced.
Component State	This property reflects the state of the underlying processing component (*i.e.*, filter) so that a clock mechanism client may determine if the underlying processing component is in the stop state, the pause state, the run state, or the data acquisition state. Again, this property functions similar to the media time in that it is dependent upon what is happening to the underlying data stream.

Function Table	This property indicates a data structure or table that allows a client to access a table of functions which represent a subset of the property set interface. Of necessity, this property is platform specific since kernel mode function calls cross filter boundaries may be handled differently by different operating systems. In the exemplary embodiment, using the NT operating system, the function table allows other kernel mode entities to have a faster interface available at the DPC level. This is contrasted with the normal method of using properties through NT kernel IRPs.

Those skilled in the art will appreciate that other properties than those listed in Table 4 or a subset of the properties listed in Table 4 may be used to make a clock mechanism property set that provides functionality according to the interconnected filter system explained herein. Furthermore, the flexible property set mechanism allows a filter developer with unique timing capabilities to extend a "required" property set by adding thereto another property set, having its own GUID, for more advanced capabilities. The set mechanism allows the flexibility to filter developers in that a minimum amount of functionality may be implemented so that the filter is compliant with the system architecture while allowing unbounded ability for the filter developer to customize and add additional capabilities.

While a property set allows other processing components, such as kernel mode filters, to query relevant time properties, clock mechanisms may also support an event set so that notifications may be sent directly to interested clock mechanism clients by way of event notification IRPs. The handle of a file object may be "registered" with the clock mechanism as part enabling an event so that the clock mechanism will know where to send event notifications. Optionally, direct function call handles may be used so that DPC level processing can occur between kernel mode entities for achieving higher performance.

Table 5 below, a number of possible notification events make up an event set that can be supported by compliant drivers. A third party controlling agent may interconnect or register those entities for receiving events based on a given filter graph topology.

TABLE 5

Clock Mechanism Events	
Event	**Description**
Interval Event	An event which is to be signalled in a cyclic manner. In the exemplary embodiment, a specific interval and specified clock time are set by a controlling agent. Beginning at the specified clock time, cyclic event notifications will be sent at the occurrence of each interval. For example, if a processing component renders video frames from a video stream at thirty frames per second, it would specify a start time and an interval of 1/30th of a second so that the event notifications would be sent at the same frequency thereby allowing the video frames to be rendered properly.
Positional Event	An event which is to be signalled at a specified clock time. For example, a processing component or filter which renders a media event stream may have the need, in some circumstances, to play notes at a specific clock time. The processing component or filter could specify a positional event notification to occur at the designated clock time so that it would know to begin rendering the note. Positional notification events could also be used to determine exactly when the clock begins to progress after being put into the run state by setting an event notification on the stopped or paused clock to be signalled at the current time.

Those skilled in the art will appreciate that other timing related events may be created for a clock mechanism. For example, events may be broken down according to the different kinds of time (*i.e.*, media time or physical time) that a clock mechanism may support. Furthermore, "hybrid" events may be envisioned by combining elements of interval events and positional events, as well as other events based on state changes.

Referring now to Figure 19A, a method for synchronizing one or more "slave" processing components or filters based on a "master" media processing time is presented. After starting at step 380, a slave processing component or filter receives the master clock media time value at step 382. Such receipt may occur by having the slave filter or component query the master clock mechanism for the media time value or the master clock mechanism may send notifications to the slave processing component or filter. Applying the

method explained by the flow chart of Figure 19A to the filter graph topology shown in Figures 17 and 18, the master clock mechanism 344 is represented by a file object which would communicate with the input pin instance 336 of the video renderer filter 332 as a slave processing component.

Note also, that the media time value used depends upon the master data stream processing. Should media rendering be paused or stopped, then time would effectively stop for synchronization purposes.

The slave processing component or filter will compare the master clock media time value to the slave data stream media time value at step 384 in order to determine how far ahead or behind the slave media processing has become. Note that media time is based upon the time interval information in the actual stream of data samples and therefor can be viewed as positional in nature. In other words, synchronization also means processing two data streams at the same relative time position within the stream.

Before ending at step 388, the slave processing component or filter will adjust slave media processing to match the master media processing time at step 386. Such adjustments may include accelerating or decelerating slave media stream processing, changing the rate of the underlying hardware processor, etc.

Decelerating media stream processing rate may be achieved by duplicating samples for processing, introducing intermittent time delays, or resampling. Accelerating media stream processing may be achieved by omitting media samples for processing, or resampling. Those skilled in the art will appreciate that numerous ways exist for adjusting the media processing rate once it is determined that a change must be made.

Note that in the example topology shown in Figures 17 and 18, input pin instance 336 would receive timing notifications for the video renderer filter 332. The video renderer filter 332 would make whatever adjustments to processing that are necessary in order to

bring the processing of the video stream of Figure 17 . in synchronization with the processing of the audio stream of Figure 18 .

Referring now to the flow chart of Figure 19, processing steps for rate matching a live or real time audio stream origination rate with a processing component or filter processing rate is shown. After beginning at step 390, physical time samples are received at the processing component or filter at step 392. Again, this can be done by querying a clock mechanism or by receiving a timing event notification. With reference to Figure 18 , processing of the live audio stream 352 which is ultimately rendered at audio renderer filter 364, the relevant processing component would be the audio renderer filter 364 as event notifications and control information are received at the input pin instance 368. Also, the clock mechanism 344 would provide the different timing information.

From the physical time samples received at step 392, the amount of data processed between those time samples can be used to compute or determine a physical time rate at step 394. The physical time rate is the rate at which the audio hardware actually renders data and represents data rendering throughput and is also known as the filter processing rate. If the source of the live data stream produces data samples at a faster origination rate than the filter processing rate, excess data occurs. If the origination rate is less than the filter processing rate, then gaps or starvation for data processing occur. In either case, performance degradation results.

The actual processing rate of the processor that originates the data samples, or ordination rate, can be determined from the media or data samples themselves. Because the media samples have time interval information in the form of convention or actual timestamp information, media time samples may be computed from the media stream at step 396. This allows the origination rate representing the processing rate of the hardware creating the media samples to be computed by taking a fixed amount of data processed and dividing this

quantity by time it took for that data to be processed as determined by the media time samples.

Before ending at step 400, the media stream processing is adjusted to match the ordination rate to the filter processing rate at step 398. Again, adjustments may occur by omitting samples for rendering, adding time delays between sample rendering, and other ways known in the art.

Referring now to the flow chart shown in Figure 19C, processing steps for using a clock mechanism to make a translation is shown. After beginning at step 402, a processing component or filter will receive a correlated time value from a clock mechanism at step 404. Note that the processing component or filter will have access to the reference time clock, such as a PC clock, that generates the reference time value portion of the correlated time value. A media time delta is computed at step 406 by subtracting the media time value from the PC clock or other reference time value as found from the correlated time value received previously. The media time delta will be used by the processing component or filter for translation and represents the variation of the media time from the PC clock time.

In order to make the translation, the processing component or filter gets a current PC clock time value (or other reference time value) at step 408. The current media time at the processing component or filter may be set or timestamp information may be translated using the current PC clock time value and the master media time delta at step 410. Since the media time delta shows the variation of the media time from PC clock time in the reference time basis, it may be simply added to the current PC clock time value in order to arrive at a proper "translated" or new time value for timestamp information. After finalizing the translation process in step 410, processing ends at step 412. If this communication were taking place between two components on a remote processor, such as a DSP, the reference time value would be the local host time rather than the PC time.

Applying the method for translation illustrated in the flow chart of Figure 19C to the interconnected kernel mode filters for processing and rendering a live audio and video stream as shown in Figures 17 and 18, it would be the input pin instance 336 on the video renderer filter 332 that would receive the correlated time from the clock mechanism 344 at step 404. Finally, it would be the video renderer filter 332 that made the computations and translation of either the timestamps on the live video stream 320 or otherwise created a translated reference time based on the media time of the clock mechanism 344.

Those skilled in the art will recognize that in a system of interconnected filters as disclosed herein, the uniform buffer allocation facilities and the timing facilities may be used together in combination for efficient processing or used separately depending implementation problem solution requirements. Furthermore, the timing facilities may be used independently of a system of interconnected filters.

Those skilled in the art will recognize that the methods of the present invention may be incorporated as computer instructions stored as a computer program code means on a computer readable medium such as a magnetic disk, CD-ROM, and other media common in the art or that may yet be developed. Furthermore, important data structures found in computer hardware memory may be created due to operation of such computer program code means.

The present invention may be embodied in other specific forms without departing from its spirit of essential characteristics. The described embodiments are to be considered in all respects only as illustrated and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

What is claimed and desired to be secured by United States Letters Patent is:

4. Brief Description of the Drawings

Figure 1 is a prior art data flow diagram showing a system of interconnected filters and drivers under the direction of a controlling agent for bringing sound data from a disk file, processing the sound data in some form, and rendering the sound data to be played through a speaker.

Figure 2 shows a system according to the present invention having the same purpose as that shown in Figure 1 to read sound data from a disk drive, process that data, and render that data to be heard on a speaker, wherein the processing filters and rendering are handled by interconnected kernel mode drivers, again under the direction of a controlling agent.

Figure 3 is a vertical relationship model showing the relationships between driver objects, device objects and file objects as created and used in an operating system.

Figures 4A, 4B and 4C are logical block diagrams of a driver object, device object, and file object, respectively, showing their logical relationship with the data structures and program code to route messages to appropriate process handling code and to validate the creation of new file objects according to the system of the present invention.

Figure 5 is a flowchart showing the initial set up of the routing and validation componentry and the processing of I/O messages by the kernel mode drivers.

Figure 6 is a flowchart showing in more detail the processing of a controlling agent, the routing and validation mechanisms, and specific create handler routines for creating a new file object.

Figure 7 is a logical diagram showing the horizontal relationship between connected filters utilizing the file object structures in an operating system to effectuate such a connection in a standardized fashion.

Figure 8 is a flowchart showing the processing steps taken by a controlling agent in user mode to create and connect the kernel mode filters or drivers of Figure 7 in order to effectuate a connection for processing I/O requests received from the controlling agent with processing continuing between different drivers (filters).

Figures 9 and 10 are logical overview diagrams of the kernel mode drivers and connections used to create a chain of kernel mode filters under the direction of a user mode controlling agent to implement a system for reading sound data from a hard drive, processing the data with the kernel mode filters, and rendering the data to be heard through a speaker.

Figure 11 is a flowchart showing the processing steps for creating the interconnected kernel mode drivers for the system shown in Figures 9 and 10.

Figures 12 and 13 illustrate the functioning of a buffer allocator mechanism. Figure 12 shows a logical arrangement and processing of the allocated buffer frames as they are passed from one processing component to another. Figure 13 illustrates a buffer allocator being represented as a file object that is a "child" of a file object representing an input pin instance in a system of interconnected kernel mode filters. Both Figures 12 and 13 illustrate the same filter graph topology.

Figure 14 shows the buffer allocation in transitions of the system illustrated in Figures 9 and 10 utilizing buffer allocators for controlling the allocation of buffer frames.

Figure 15 is a flow chart showing the processing steps for bringing data from a disk driver through a chain of interconnected kernel mode filters and rendering the data on sound

processing hardware specifically showing the operation of buffer allocators and the actual data transferring between buffers for the system shown in Figure 1 .

Figure 16 is a logical block diagram illustrating how two live data streams can be synchronized to a single master clock mechanism.

Figures 17 and 18 are logical block diagrams showing the live audio system of Figure 16 as implemented using the interconnected filter system explained in detail in connection with Figure 14. Figure 17 represents the live video stream rendering filters which receives a master clock synchronization signal, and Figure 18 illustrates the live audio rendering system that has a master clock mechanism for synchronizing both streams together and for rate matching the live audio data with the actual audio rendering hardware.

Figures 19A-19C illustrate how a clock mechanism may be used to synchronize data processing of multiple data streams, rate match a stream of data with the physical capabilities of a hardware renderer, and translate one time base to another using a common time base. Figure 19A is a flow chart showing the synchronization processing steps. Figure 19B is a flow chart showing the rate matching processing steps. Figure 19C is a flow chart showing the translation processing steps.

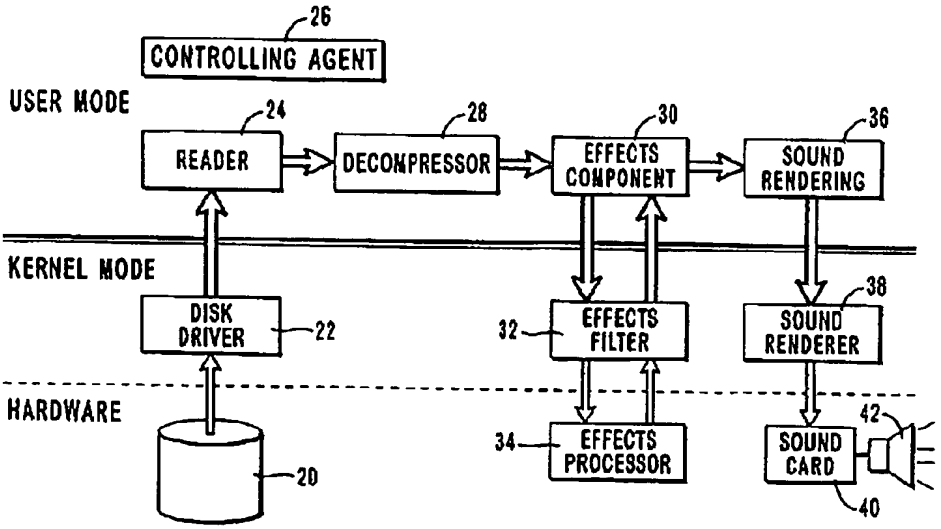


FIG. 1

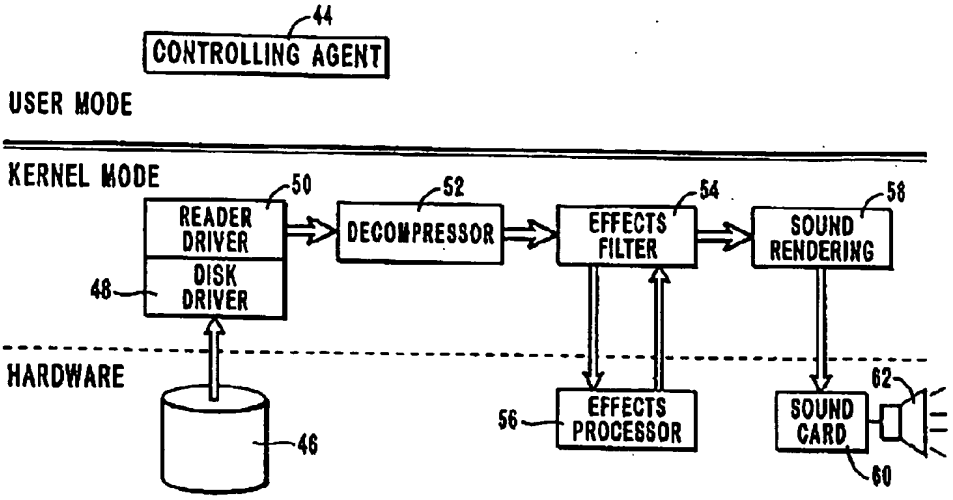

44
CONTROLLING AGENT
USER MODE
KERNEL MODE
50
READER DRIVER
DISK DRIVER
48
52
DECOMPRESSOR
54
EFFECTS FILTER
58
SOUND RENDERING
HARDWARE
46
56
EFFECTS PROCESSOR
62
SOUND CARD
60

FIG. 2

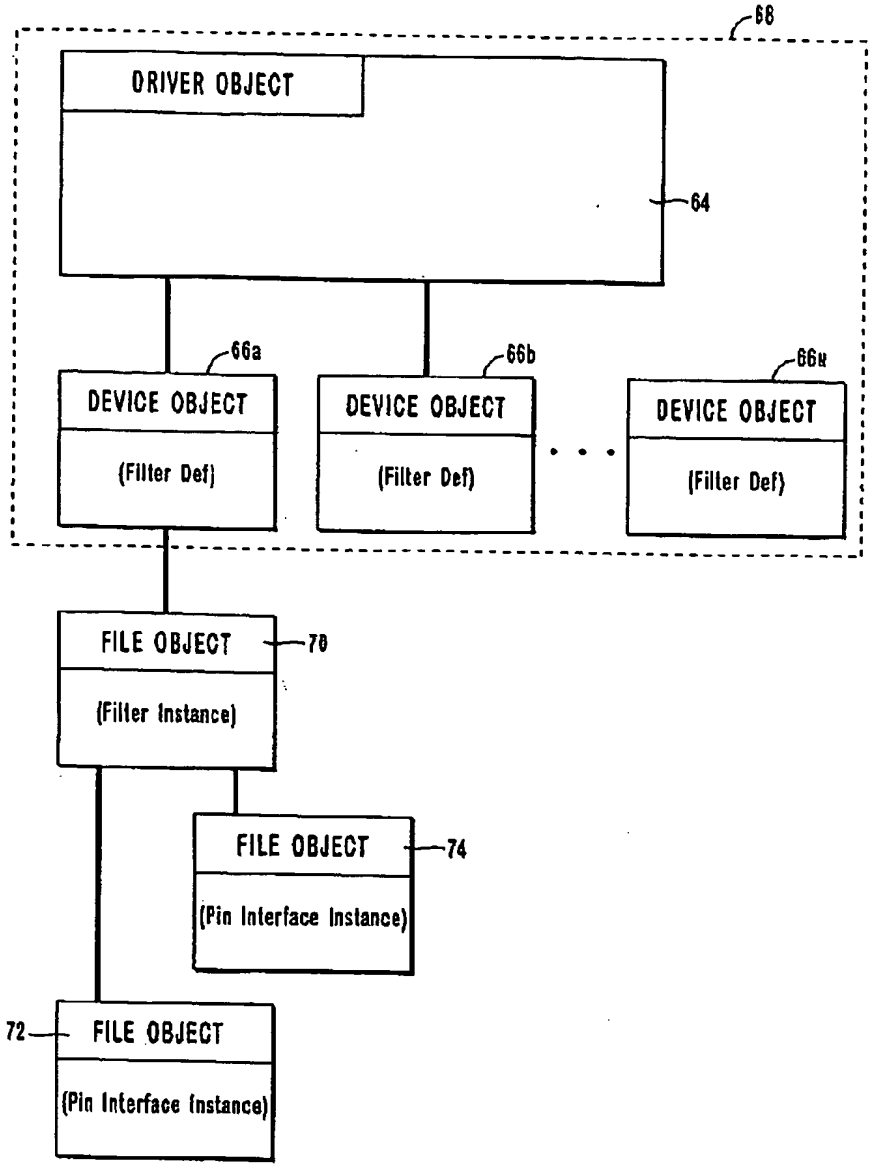
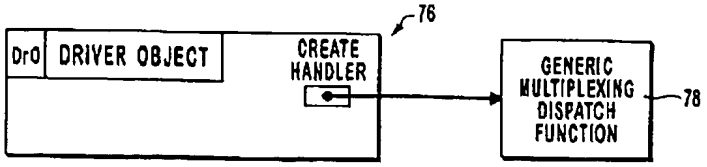
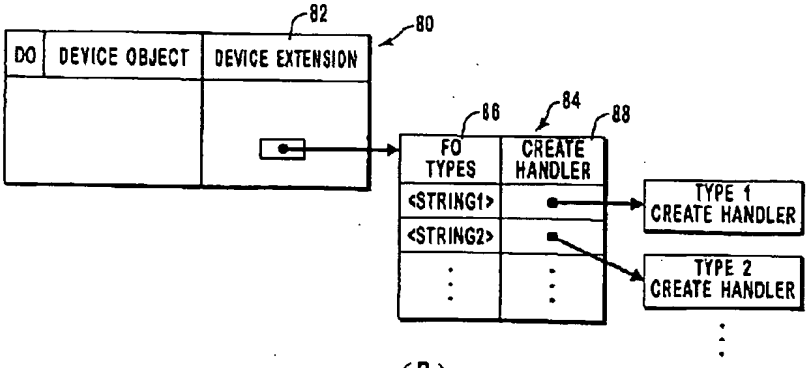

68
DRIVER OBJECT
64
66a
DEVICE OBJECT
(Filter Def)
66b
DEVICE OBJECT
(Filter Def)
66n
DEVICE OBJECT
(Filter Def)
FILE OBJECT
(Filter Instance)
70
FILE OBJECT
(Pin Interface Instance)
74
72
FILE OBJECT
(Pin Interface Instance)

FIG. 3


76
DrO
DRIVER OBJECT
CREATE HANDLER
GENERIC MULTIPLEXING DISPATCH FUNCTION
78

(A)


82
80
DO
DEVICE OBJECT
DEVICE EXTENSION
86
84
88
FO TYPES
CREATE HANDLER
<STRING1>
<STRING2>
TYPE 1 CREATE HANDLER
TYPE 2 CREATE HANDLER

(B)

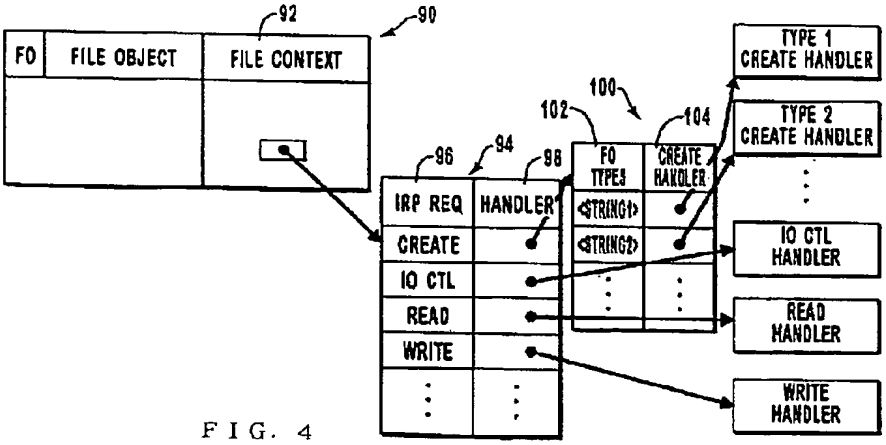

92
90
FO
FILE OBJECT
FILE CONTEXT
100
102
104
96
94
98
IRP REQ
HANDLER
CREATE
IO CTL
READ
WRITE
FO TYPES
CREATE HANDLER
<STRING1>
<STRING2>
TYPE 1 CREATE HANDLER
TYPE 2 CREATE HANDLER
IO CTL HANDLER
READ HANDLER
WRITE HANDLER

FIG. 4

(C)

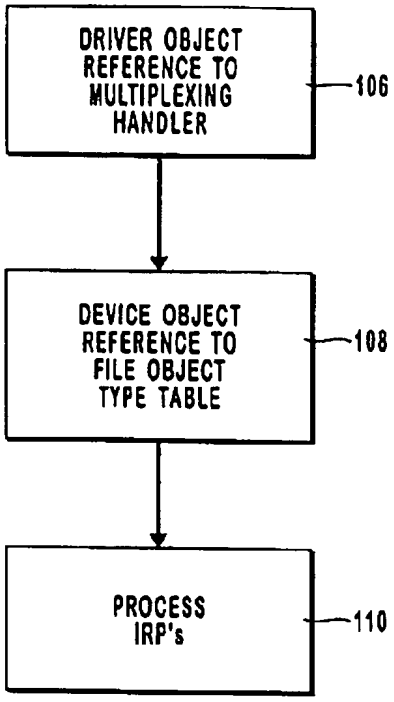

DRIVER OBJECT REFERENCE TO MULTIPLEXING HANDLER
106
DEVICE OBJECT REFERENCE TO FILE OBJECT TYPE TABLE
108
PROCESS IRP's
110

FIG. 5

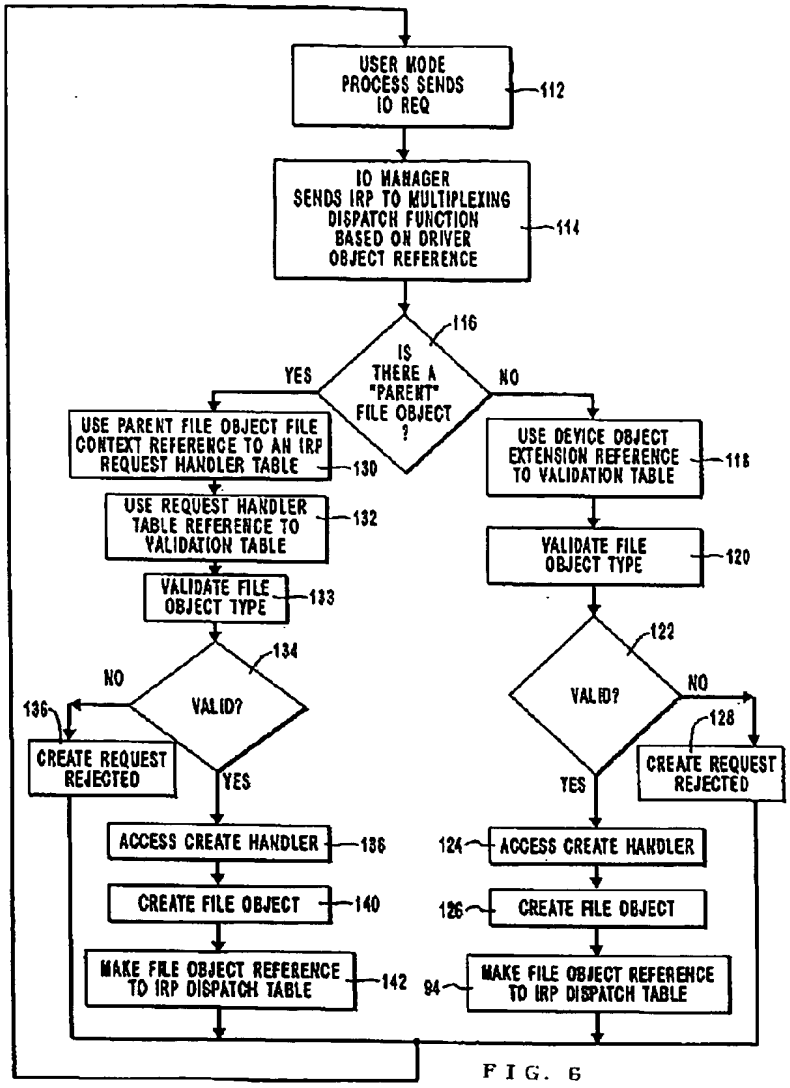

USER MODE PROCESS SENDS IO REQ
112
IO MANAGER SENDS IRP TO MULTIPLEXING DISPATCH FUNCTION BASED ON DRIVER OBJECT REFERENCE
114
IS THERE A "PARENT" FILE OBJECT ?
116
YES
NO
USE PARENT FILE OBJECT FILE CONTEXT REFERENCE TO AN IRP REQUEST HANDLER TABLE
130
USE REQUEST HANDLER TABLE REFERENCE TO VALIDATION TABLE
132
VALIDATE FILE OBJECT TYPE
133
VALID?
134
NO
136
CREATE REQUEST REJECTED
YES
ACCESS CREATE HANDLER
136
CREATE FILE OBJECT
140
MAKE FILE OBJECT REFERENCE TO IRP DISPATCH TABLE
142
USE DEVICE OBJECT EXTENSION REFERENCE TO VALIDATION TABLE
118
VALIDATE FILE OBJECT TYPE
120
VALID?
122
NO
128
CREATE REQUEST REJECTED
YES
124
ACCESS CREATE HANDLER
126
CREATE FILE OBJECT
94
MAKE FILE OBJECT REFERENCE TO IRP DISPATCH TABLE

FIG. 6

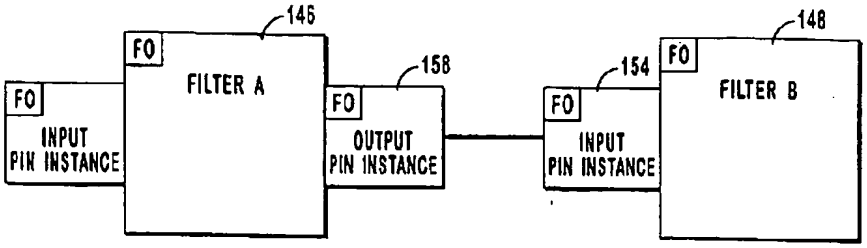

146
FO
FILTER A
FO
INPUT PIN INSTANCE
158
FO
OUTPUT PIN INSTANCE
154
FO
INPUT PIN INSTANCE
148
FO
FILTER B

FIG. 7

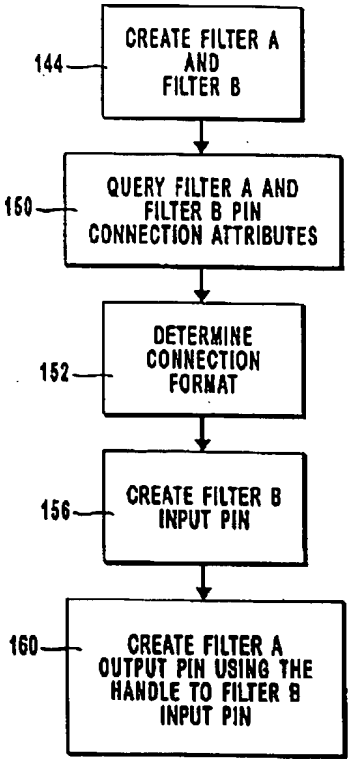

144
CREATE FILTER A AND FILTER B
150
QUERY FILTER A AND FILTER B PIN CONNECTION ATTRIBUTES
152
DETERMINE CONNECTION FORMAT
156
CREATE FILTER B INPUT PIN
160
CREATE FILTER A OUTPUT PIN USING THE HANDLE TO FILTER B INPUT PIN

FIG. 8

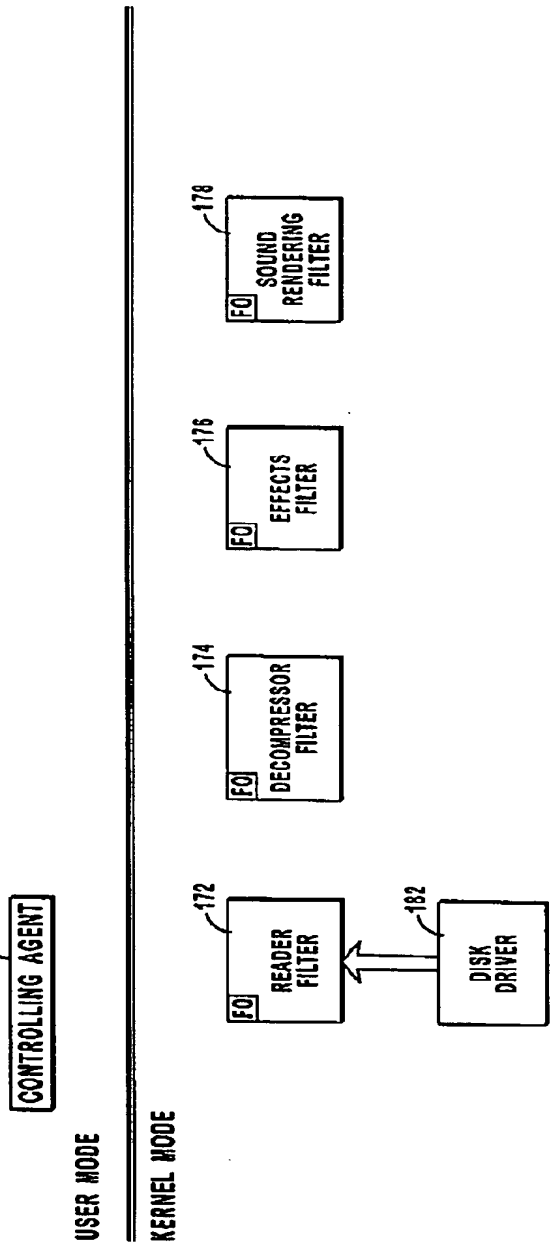

170
CONTROLLING AGENT
USER MODE
KERNEL MODE
172
FO
READER FILTER
182
DISK DRIVER
174
FO
DECOMPRESSOR FILTER
176
FO
EFFECTS FILTER
178
FO
SOUND RENDERING FILTER

FIG. 9

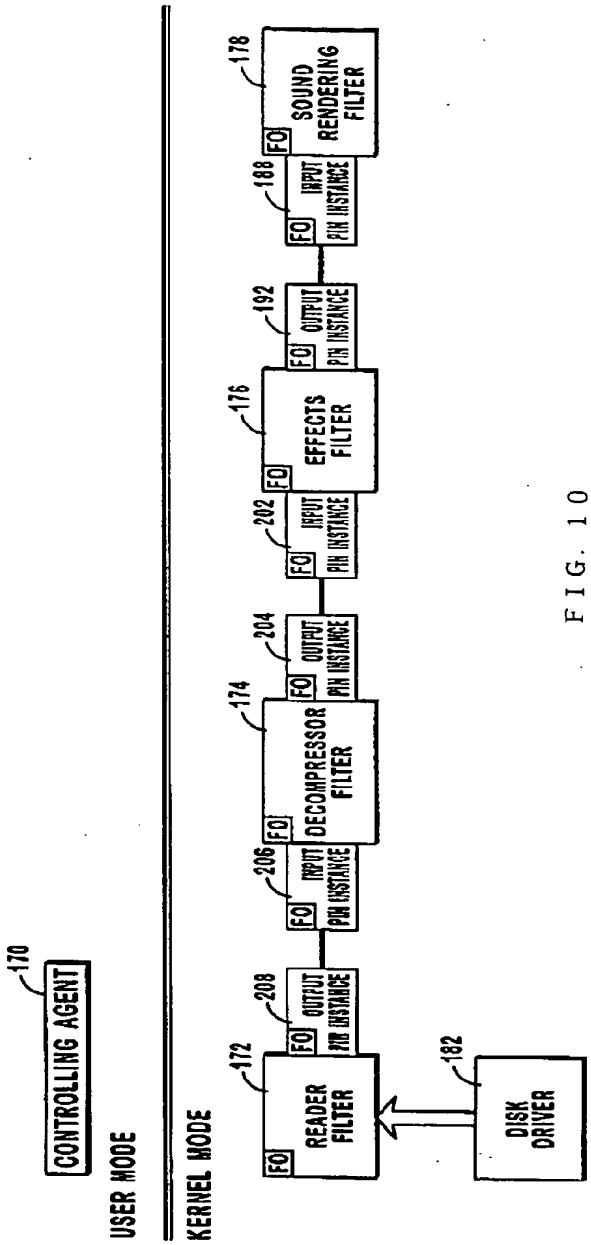

170
CONTROLLING AGENT
USER MODE
KERNEL MODE
172
FO
READER FILTER
208
FO
OUTPUT PIN INSTANCE
182
DISK DRIVER
206
FO
INPUT PIN INSTANCE
174
FO
DECOMPRESSOR FILTER
204
FO
OUTPUT PIN INSTANCE
202
FO
INPUT PIN INSTANCE
176
FO
EFFECTS FILTER
192
FO
OUTPUT PIN INSTANCE
188
FO
INPUT PIN INSTANCE
178
FO
SOUND RENDERING FILTER

FIG. 10

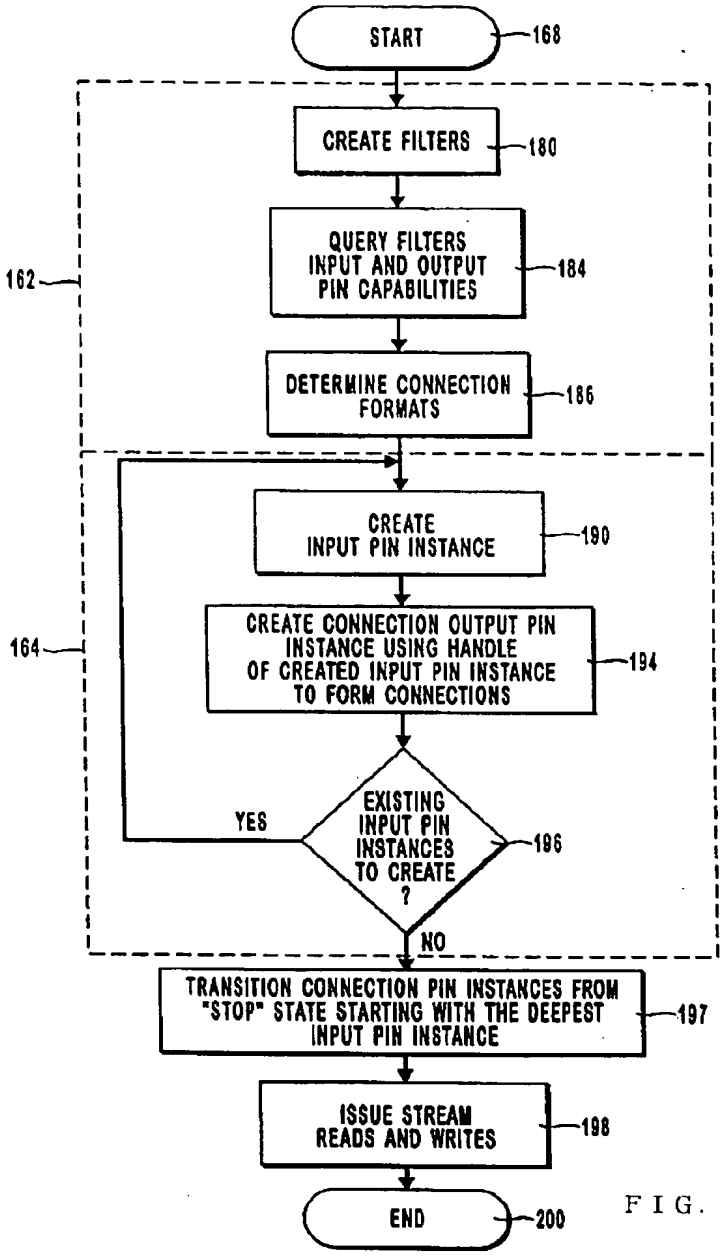

START
168
CREATE FILTERS
180
QUERY FILTERS INPUT AND OUTPUT PIN CAPABILITIES
184
162
DETERMINE CONNECTION FORMATS
186
CREATE INPUT PIN INSTANCE
190
CREATE CONNECTION OUTPUT PIN INSTANCE USING HANDLE OF CREATED INPUT PIN INSTANCE TO FORM CONNECTIONS
194
164
EXISTING INPUT PIN INSTANCES TO CREATE ?
196
YES
NO
TRANSITION CONNECTION PIN INSTANCES FROM "STOP" STATE STARTING WITH THE DEEPEST INPUT PIN INSTANCE
197
ISSUE STREAM READS AND WRITES
198
END
200

FIG. 11

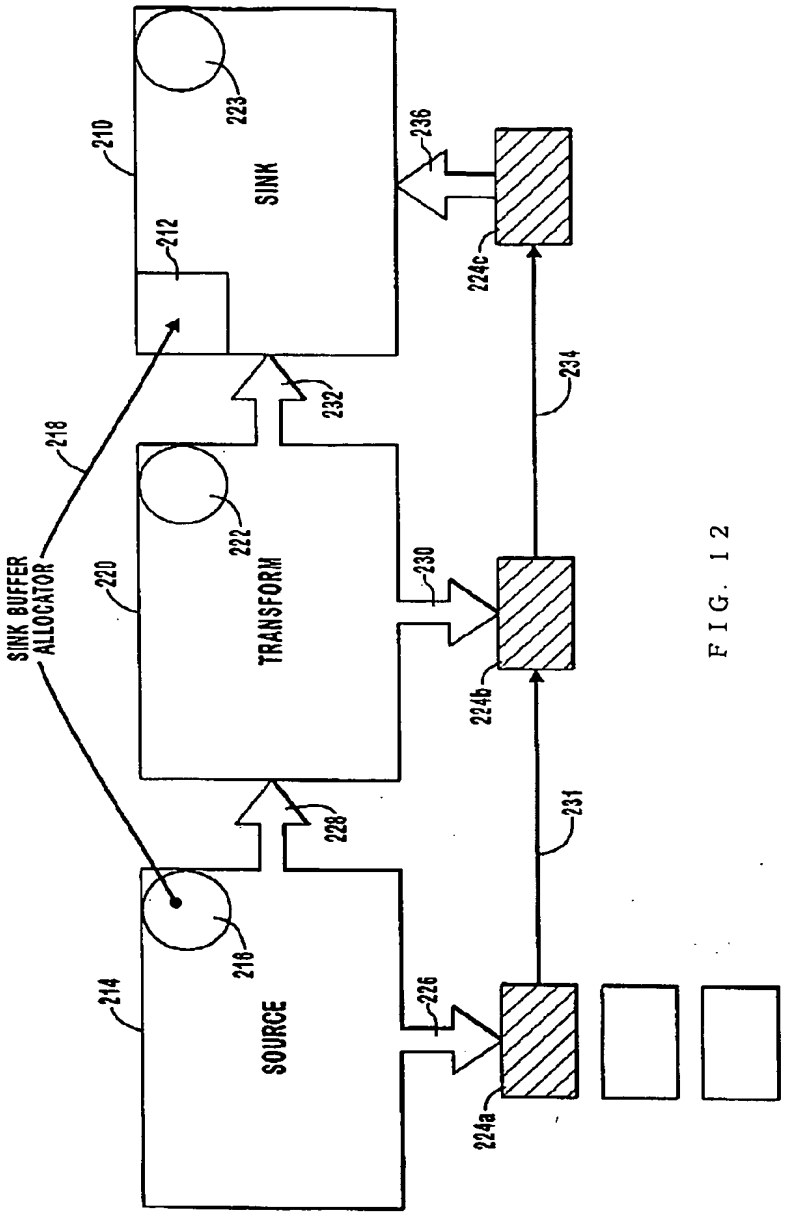

SINK BUFFER ALLOCATOR
218
210
SINK
223
212
236
224c
234
232
220
TRANSFORM
222
230
224b
231
228
214
SOURCE
216
226
224a

FIG. 12

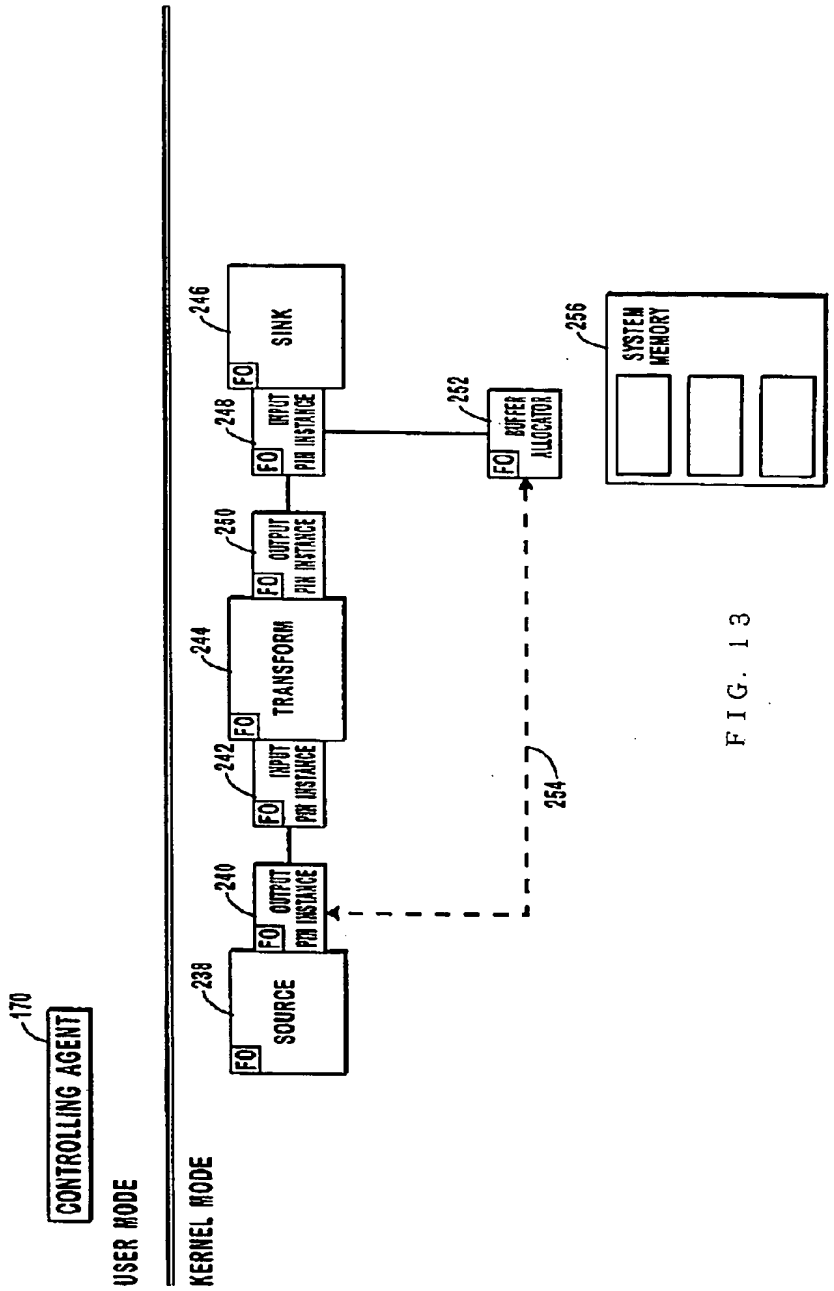

170
CONTROLLING AGENT
USER MODE
KERNEL MODE
238
SOURCE
FO
240
OUTPUT PIN INSTANCE
242
INPUT PIN INSTANCE
244
TRANSFORM
246
SINK
248
INPUT PIN INSTANCE
250
OUTPUT PIN INSTANCE
252
BUFFER ALLOCATOR
254
256
SYSTEM MEMORY

FIG. 13

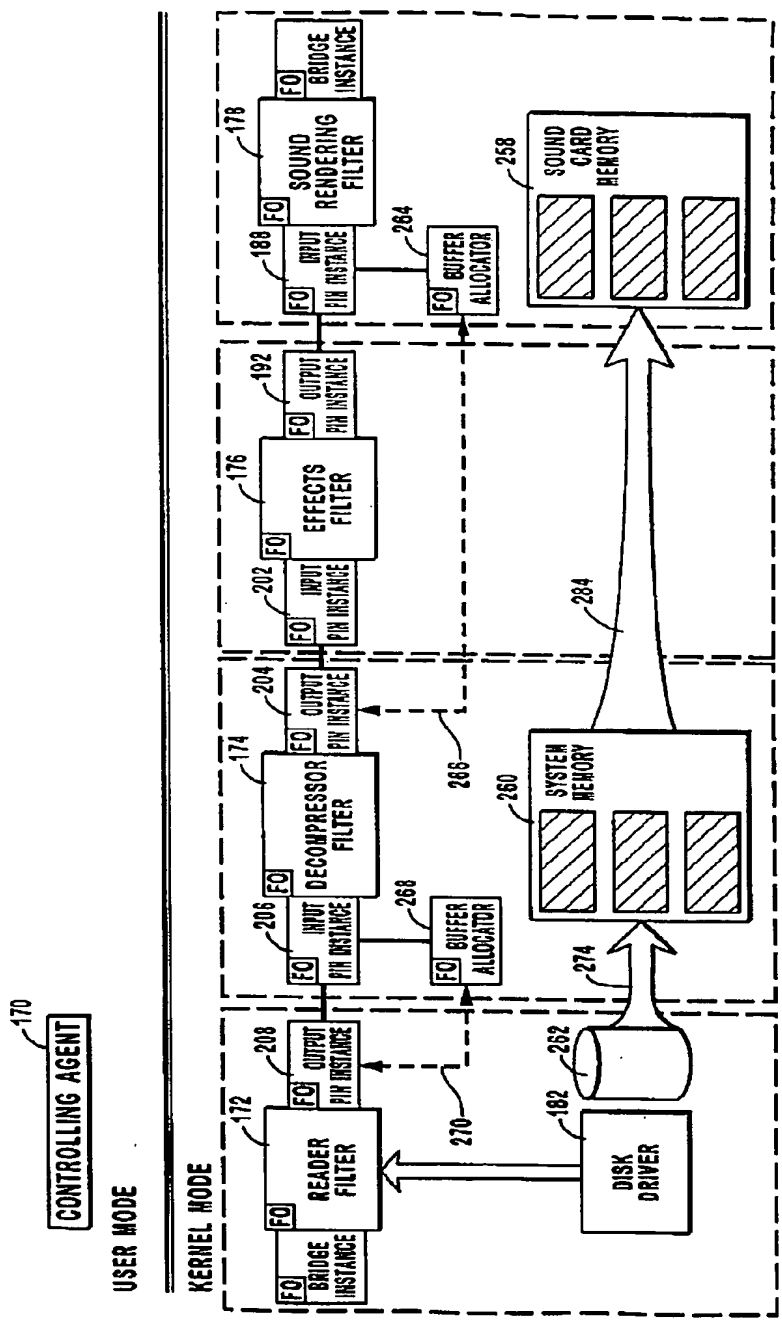

170
CONTROLLING AGENT
USER MODE
KERNEL MODE
FO BRIDGE INSTANCE
172
READER FILTER
208
OUTPUT PIN INSTANCE
270
182
DISK DRIVER
262
274
206
INPUT PIN INSTANCE
268
BUFFER ALLOCATOR
174
DECOMPRESSOR FILTER
204
OUTPUT PIN INSTANCE
266
260
SYSTEM MEMORY
202
INPUT PIN INSTANCE
176
EFFECTS FILTER
192
OUTPUT PIN INSTANCE
284
188
INPUT PIN INSTANCE
178
SOUND RENDERING FILTER
BRIDGE INSTANCE
264
BUFFER ALLOCATOR
258
SOUND CARD MEMORY

FIG. 14

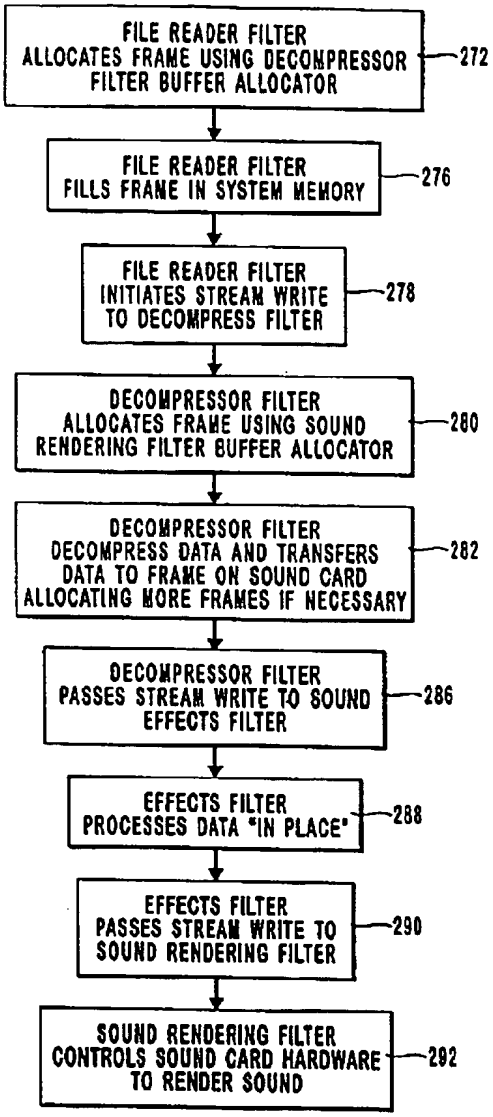

FILE READER FILTER ALLOCATES FRAME USING DECOMPRESSOR FILTER BUFFER ALLOCATOR
272
FILE READER FILTER FILLS FRAME IN SYSTEM MEMORY
276
FILE READER FILTER INITIATES STREAM WRITE TO DECOMPRESS FILTER
278
DECOMPRESSOR FILTER ALLOCATES FRAME USING SOUND RENDERING FILTER BUFFER ALLOCATOR
280
DECOMPRESSOR FILTER DECOMPRESS DATA AND TRANSFERS DATA TO FRAME ON SOUND CARD ALLOCATING MORE FRAMES IF NECESSARY
282
DECOMPRESSOR FILTER PASSES STREAM WRITE TO SOUND EFFECTS FILTER
286
EFFECTS FILTER PROCESSES DATA "IN PLACE"
288
EFFECTS FILTER PASSES STREAM WRITE TO SOUND RENDERING FILTER
290
SOUND RENDERING FILTER CONTROLS SOUND CARD HARDWARE TO RENDER SOUND
292

FIG. 15

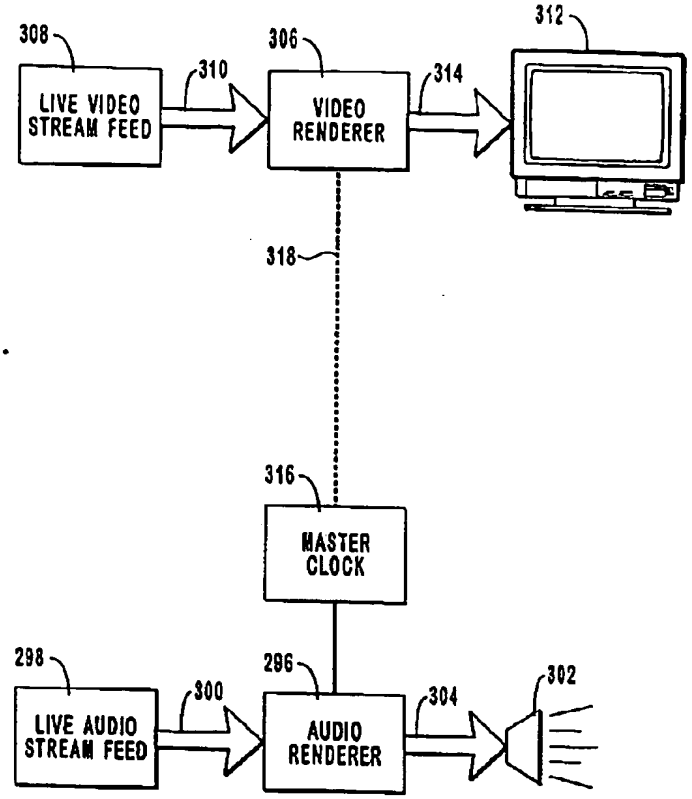

308
LIVE VIDEO STREAM FEED
310
306
VIDEO RENDERER
314
312
318
316
MASTER CLOCK
298
LIVE AUDIO STREAM FEED
300
296
AUDIO RENDERER
304
302

FIG. 16

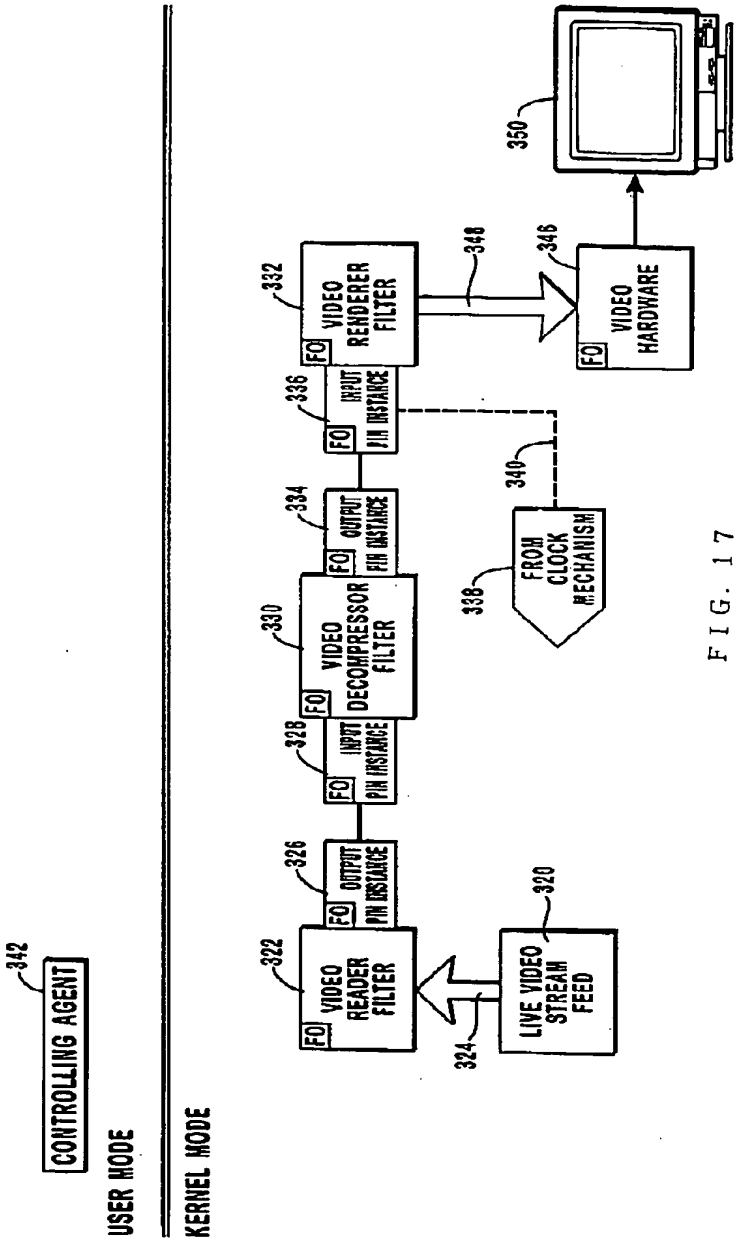

342
CONTROLLING AGENT
USER MODE
KERNEL MODE
322
VIDEO READER FILTER
326
OUTPUT PIN INSTANCE
328
INPUT PIN INSTANCE
330
VIDEO DECOMPRESSOR FILTER
334
OUTPUT PIN INSTANCE
336
INPUT PIN INSTANCE
332
VIDEO RENDERER FILTER
320
LIVE VIDEO STREAM FEED
324
338
FROM CLOCK MECHANISM
340
348
346
VIDEO HARDWARE
350

FIG. 17

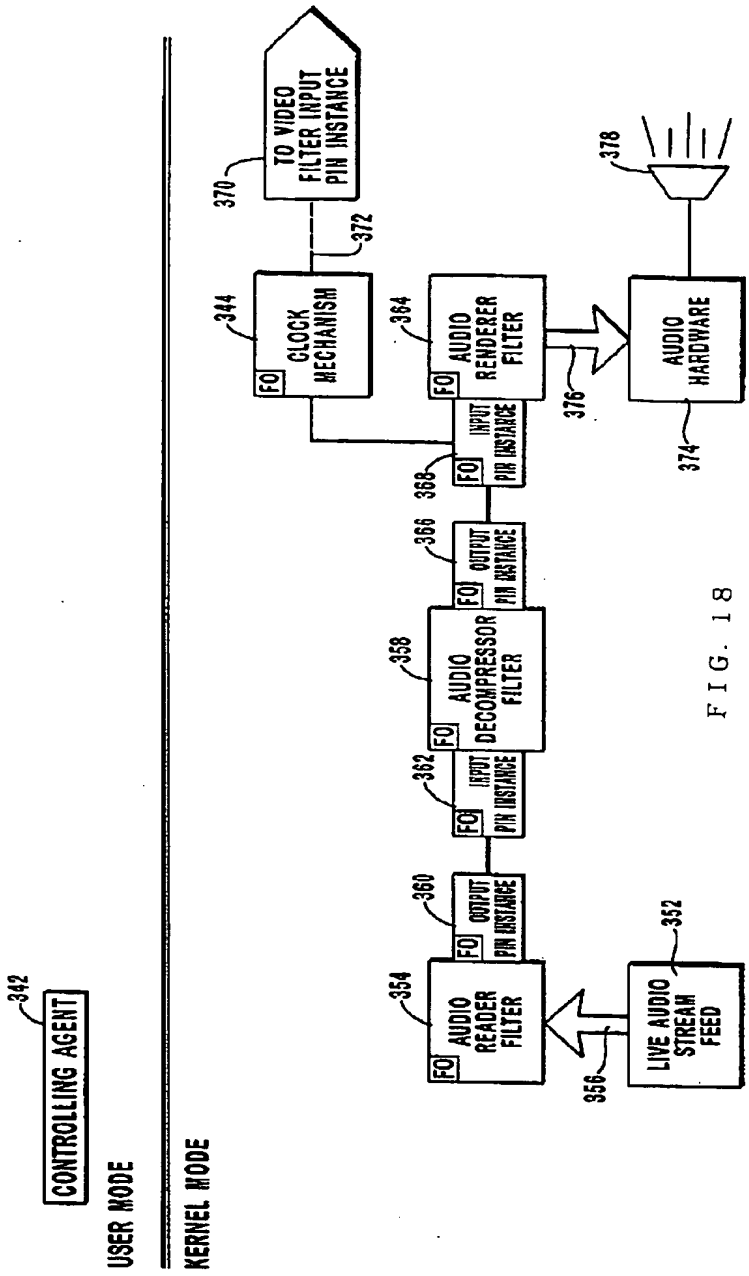

342
CONTROLLING AGENT
USER MODE
KERNEL MODE
370
TO VIDEO FILTER INPUT PIN INSTANCE
372
344
FO
CLOCK MECHANISM
364
FO
AUDIO RENDERER FILTER
378
AUDIO HARDWARE
376
374
368
FO
INPUT PIN INSTANCE
366
FO
OUTPUT PIN INSTANCE
358
FO
AUDIO DECOMPRESSOR FILTER
362
FO
INPUT PIN INSTANCE
360
FO
OUTPUT PIN INSTANCE
354
FO
AUDIO READER FILTER
352
LIVE AUDIO STREAM FEED
356

FIG. 18

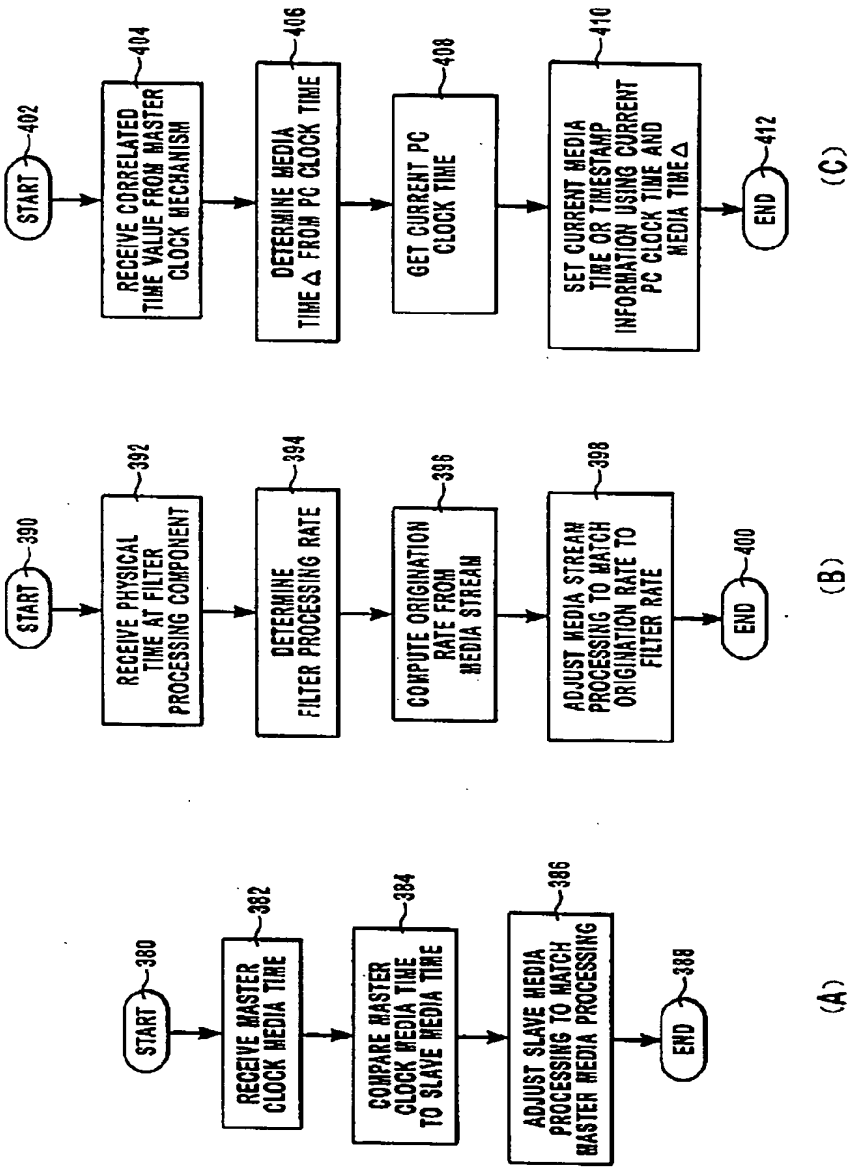

START
402
RECEIVE CORRELATED TIME VALUE FROM MASTER CLOCK MECHANISM
404
DETERMINE MEDIA TIME Δ FROM PC CLOCK TIME
406
GET CURRENT PC CLOCK TIME
408
SET CURRENT MEDIA TIME OR TIMESTAMP INFORMATION USING CURRENT PC CLOCK TIME AND MEDIA TIME Δ
410
END
412
(C)
START
390
RECEIVE PHYSICAL TIME AT FILTER PROCESSING COMPONENT
392
DETERMINE FILTER PROCESSING RATE
394
COMPUTE ORIGINATION RATE FROM MEDIA STREAM
396
ADJUST MEDIA STREAM PROCESSING TO MATCH ORIGINATION RATE TO FILTER RATE
398
END
400
(B)
START
380
RECEIVE MASTER CLOCK MEDIA TIME
382
COMPARE MASTER CLOCK MEDIA TIME TO SLAVE MEDIA TIME
384
ADJUST SLAVE MEDIA PROCESSING TO MATCH MASTER MEDIA PROCESSING
386
END
388
(A)

FIG. 19

１．Abstract

A method and computer program product for synchronizing processing between two or more data streams (*e.g.*, video and sound input) and for rate matching between two different hardware clocks that may drift with respect to one another (*e.g.*, an originating clock represented in a timestamped data stream versus a clock actually rendering the data) in a system of interconnected software drivers running in kernel mode. The present invention overcomes the coordination complexity and inaccuracies in the prior art by providing a clocking mechanism in a system wherein multiple drivers having input and output connection pin instances are chained together. The clocking mechanism synchronizes between data streams by providing a master clock on an input pin instance of a driver that is used to synchronize with other input pin instances on other drivers and "slave" clocks. Synchronization is achieved through event notification or stream position queries so that corresponding frames of data in separate streams are rendered together (*e.g.*, video frames with corresponding sound track). Rate matching is achieved through monitoring a physical clock progression in comparison with a series of data stream timestamps thereby allowing adjustments to match the different clock rates. A common physical clock (*e.g.*, PC clock) can be used as a reference for a component to translate a particular clock time to a time shared by all components with a minimum of error.

２．Representative Drawing

ＦＩＧ．２